EXHIBIT F

Input Processing Unit High Level Design

Revision 1.38

June 13, 2002

Bob Sefton
Qian Yao

Revision History

Revision	Date	Author	Description of Changes
0.10	April 17, 2001	Pramod Argade	Created template
0.20	May 14, 2001	Bob Sefton	First draft of SPI-4 Receiver sections
0.30	May 25, 2001	Bob Sefton	Removed SPI-4 core sections; completed Event Queue and Input Header Buffer sections; renamed to Input Processing Unit HLD
0.40	June 12, 2001	Bob Sefton	First draft of complete document
1.00	June 15, 2001	Bob Sefton	Changes from HLD review: - Changed references to six Input Processors to four. - Fixed typos. - Renamed IPU flow-through mode to cut-through mode. - Added unresolved issues to Open Issues list.
1.10	August 7, 2001	Bob Sefton	Major update: - Added IPU "warm reconfiguration" requirements to Input Receive Module. (Section 3.1.5) - Doubled Read Command Queue depth to 1024. (Section 5.1) - Changed Dispatcher interface to pre-read the first two quad-words, delete the start signal, and add the frame repeat signal. (Section 3.2.4) - Added requirement to delay release commands from the PPs to the Scratchpad until the corresponding frame is complete. (Section 3.2.3) - Added CRC-32 generation. (Section 3.1.5) - Updated Packet Processor interfaces to reflect new PP. (Table 4 and Table 6) - Changed assumed core clock frequency to 266MHz. (Multiple instances) - Added requirement to zero out invalid bytes in last word of a frame to SP & PPs. (Section 3.1.4.8) - Added SPI-4 framing errors, invalid port addresses, and SPI-4 Rx FIFO overflow to IPU error handling. (Section 3.3) - Removed MaxBurst1 & MaxBurst2 from queue state. (Table 1) - Added PP code entry point field to queue state (Table 1) and PP interface (Table 4). - Added signal tables for some IPU internal interfaces. (Table 3 and Table 5) - Added SPI-4 "full frame" mode support. (Section 3.1.4.8) - Added reference to MMC HLD to define management interface.
1.15	November 9, 2001	Bob Sefton Qian Yao	- Updated Scratchpad interface signal list (Table 11). - Updated Dispatcher interface signal list (Table 12). - Stated that the IPU only has one MMC interface (section 4.4). - Updated the IPU register map and register definitions (chapter 4). - Updated the IPU open issues/action items list (chapter 6).
1.20	November 21, 2001	Bob Sefton	Register map update only. Added several registers and all statistical counters are now defined.

1.30	January 14, 2002	Bob Sefton Qian Yao	Major update to reflect numerous architectural changes and incorporate fixes to errors noted in previous revisions. Change summary: - Updated register map and register definitions to fix errors and incorporate changes since last release. - Updated all sections and diagrams to reflect final IPU architecture. - Rearranged and renamed sections to better match standard Astute HLD template. - Abandoned the term "Input Processor" and introduced the term "Logical Processor" (LP) to identify the two independent processing elements inside the Packet Processor. - Added Expected Performance section. - Added Design Rationale section. - Added Initialization section. - Updated Open Issues section.
1.31	January 28, 2002	Bob Sefton	- PR 415: Stated clearly that the IPU minimum frame size is 33 bytes and that the min_frm_size field in the IRM_CONTROL register must be set no smaller than 2.
1.32	February 6, 2002	Bob Sefton	- PR-476: Added error robustness to the Frame Forwarder (section 3.1.6). - PR-498: Added IRM_RCQ_WRITE register to register map (section 4.5.25). - PR-499: Added port_queue_overflow flag to IRM_INT register (section 4.5.9). - PR-502: Fixed several typos identified by Dan Miller. - Added section describing IPU self test capabilities (section 4.7).
1.33	February 7, 2002	Bob Sefton	- PR-476: Added prevention of port queue overflow in IRM. - PR-519: Fixed FF_STATUS reg definition (section 4.5.27). - PR-528: Renamed IPU interrupt and interrupt enable regs, and added Frame Forwarder error to them (sections 4.5.31 and 4.5.32). - Added new IRM and Frame Forwarder error detection and actions to Table 8. - Renamed and updated section 3.1.4.7 on IRM error handling. - Updated Requirements in section 1.2. - Moved References section from Chapter 8 to section 1.1.1). - Renamed section 4.7 to IPU Diagnostic Support Features.
1.34	February 19, 2002	Bob Sefton	- PR-571: Fix missing EOPs in CRC Engine and update Table 8 to include CRC Engine error detection. - PR-595: Changed register names to match Pegasus HRM. - PR-612: Updated Table 3. - PR-651: Removed jabber frame detection in FF from Table 8. - PR-652: Added CRC Engine missing eop error to IPU_INT reg. - PR-653: Added Table 10: Input Processing Unit/SPI-4 Core Interface Signals.
1.35	February 25, 2002	Bob Sefton	- PR-572: Added description of small-burst mode (section 3.1.4.9). - PR-668: Expanded IPU_INT_ENA register to include a separate enable bit for each interrupt in the IPU_INT register. - Clarification of SPI-4 burst errors in section 3.1.4.7.
1.36	April 4, 2002	Bob Sefton	- PR-1090: Fixed Table 13.
1.37	April 22, 2002	Bob Sefton	- PR-1163: PP interface changes to handle frame errors. - General cleanup and clarification where needed.
1.38	June 13, 2002	Qian Yao	- PR-1414: Added IPU Throughput Summary section - Fixed typo from STARVING TO SATISFIED in Section 3.1.4.9

Table of Contents

List of Figures

List of Tables

1 Introduction

This document describes the high-level operations of the Input Processing Unit (IPU). The IPU is responsible for receiving, processing, and forwarding frames from the SPI-4 module. It presents processed information (extracted from frame headers) to the Dispatcher in the form of Input Events and forwards complete frames to the Scratchpad.

The IPU is instantiated twice in the ACP system, once each at the NetIn and HostIn interfaces.

1.1 Related Documents

Document	Revision	Author
Content Processor Architectural Overview	0.41	Fazil Osman
Packet Processor High Level Design	latest	Octera
Scratchpad High Level Design	latest	Charles Kaseff
Dispatcher High Level Design	latest	Simon Knee
Host API High Level Design	latest	Billy Oostra
MMC High Level Design	latest	Nitesh Mehta
SPI-4 Module HLD	latest	Bob Sefton

1.1.1 Additional References

[1] System Packet Interface Level 4 (SPI-4) Phase 2: OC-192 System Interface for Physical and Link Layer Devices. Implementation Agreement: OIF-SPI4-02.0, Optical Internetworking Forum, Fremont, CA, January 2001 (or latest version).

[2] SPI-4 Product Specification. SPI-4 ASIC core data sheet, Silicon Logic Engineering, Eau Claire, WI, October 22, 2001 (or latest version).

1.2 Requirements

The following IPU features are required:

- **Number of SPI-4 ports (full_frame mode OFF)**: When full_frame mode is disabled the IPU supports up to 64 SPI-4 ports. The IPU Reassembly Buffer must be configured with a separate receive queue for each SPI-4 port.
- **Number of SPI-4 ports (full_frame mode ON)**: When full_frame mode is enabled the IPU supports up to 256 SPI-4 ports (see section 3.1.4.8). In this mode the IPU uses a single receive queue to buffer all traffic, so frame reassembly is not possible and only complete frames/packets are allowed to cross the SPI-4 bus.
- **Minimum and Maximum frame sizes**: The minimum system frame size is 33 bytes. All frames <= 16 bytes are discarded by the IPU automatically, but the IPU must be configured correctly to discard frames from 17-32 bytes. It is critical that the min_frm_size field in the IRM_CONTROL register be set no smaller than two. The maximum IPU frame size is 16256 bytes. All frames larger than this will be truncated and forwarded with an error flag. The IPU also supports configurable minimum and maximum frame sizes within the 33-16256 byte limits.
- **Frame forwarding performance**: The IPU must be able to forward packets at the rate they are received from SPI-4 in the absence of downstream backpressure. Note that SPI-4 throughput may be limited in some configurations due to Reassembly Buffer size limitations. (See section 3.6.1.)
- **Header processing performance**: The IPU Packet Processors must collectively be able to process frame headers at the rate required to meet current ACP performance metrics. (See PP HLD.)
- **Frame processing order**: Frames received from a single SPI-4 port must be processed in the order received. There is no requirement to preserve frame order across ports.
- **Error Handling**: The IPU must ensure the following:
  - All frames forwarded to the Scratchpad and Packet Processor are correctly framed with start and end flags.

- All frames forwarded with known errors must be tagged with an error flag.
- All frames forwarded must be at least three quad-words in length.
- Frames longer than 16256 bytes must be truncated, flagged with an error flag, and the remainder discarded.
- Whenever possible, invalid data should be framed as a valid frame, tagged with an error flag and forwarded rather than discarded.
- All error conditions must set an interrupt status flag.

- **Detection of hung SPI-4 ports**: When the IPU Reassembly Buffer is in "cut-through" mode (see section 3.1.4.3) a timer will measure the time between SPI-4 bursts on the cut-through port. The cut-through timer will have a programmable range up to approximately 60 seconds and each port will have an independent timeout value.
- **Action for hung SPI-4 ports**: When the cut-through timer expires the IPU will truncate the current frame and set an error flag to the Scratchpad so a protocol core can drop the frame. Any further data on the "hung" port will be discarded up through the next frame end. After the next frame end any future traffic on the affected port will be processed normally. Note that if the frame end for the truncated frame never arrives, then the first "good" frame after the timeout will be dropped. This is a result of the fact that the IPU must see a frame end to exit the discard state.
- **Warm reconfiguration**: The IPU must be able to be shut down "gracefully", reconfigured, and restarted without impacting the rest of the chip. Graceful shutdown means no partial frames can be sent to the Scratchpad unless the error bit is set, and the state of the Event Sequence Number must be preserved during the process.
- **Frame release commands**: The Packet Processors can only release frames <= 256 bytes, and only if the error flag for the frame is not set. The IPU will ignore PP release commands for all other frames and will force the SP_DATA bit in the event generated by the PP to '1' so that a Protocol Core can make the release decision.
- **CRC generation**: The IPU must be able to calculate up to two CRCs per input frame. Two CRC polynomials (iSCSI and FCIP) must be supported and be selectable on a per-frame basis. A CRC control header must be the first qword of frames that require CRC processing.
- **SPI-4 full-frame mode**: As discussed in detail later in this document, the IPU Reassembly Buffer is not large enough to perform frame reassembly in the presence of large frames and/or many SPI-4 ports. To improve throughput performance under these conditions the IPU will provide a "full-frame" mode in which the SPI-4 interface passes complete frames rather than interleaved bursts. Full-frame mode abides by the SPI-4 interface protocol but requires a non-standard SPI-4 configuration.

1.3 Overview

Figure 1 shows a block diagram of the IPU. The IPU consists of three main sections:

- Input Receive Module
- CRC Engine + Frame Forwarder
- Input Processing Module

For details of the SPI-4 Module and the SPI-4 bus please refer to references [1] and [2] (see section 1.1.1), and the SPI-4 Module HLD.

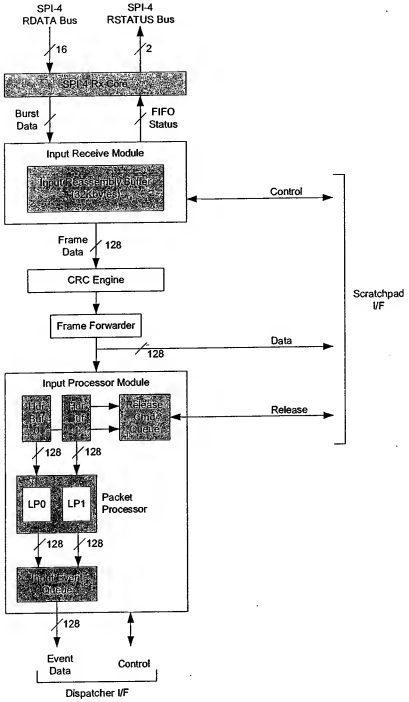

Figure 1: Input Processing Unit Block Diagram

2 IPU Functional Units

2.1 Input Receive Module

Figure 2 is a block diagram of the Input Receive Module (IRM). The following paragraphs provide a brief overview of the main IRM components. For more details on module operation, please refer to section 3.1.

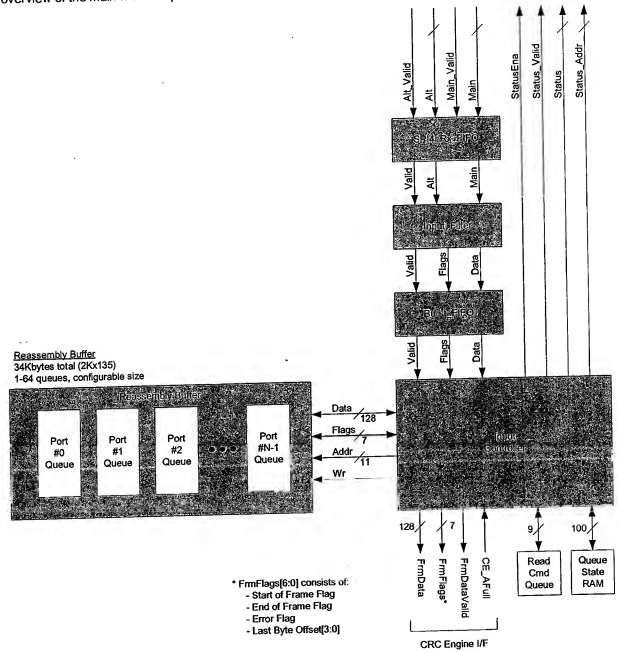

Figure 2: Input Receive Module Block Diagram

2.1.1 SPI-4 Rx FIFO

The SPI-4 Module has two data output ports: Main and Alt. The Main data port is 64 bits wide and the Alt is 32 bits wide. Each port has its own associated flags and data valid signal. The SPI-4 Module has no input buffering, and under some conditions the Main data port is not able to pass all traffic. The Alt port provides an overflow path when this occurs. The SPI-4 Rx FIFO (64x124) buffers Main and Alt data and flags, then merges them into a single 64-bit bus with flags to the Input Filter.

2.1.2 Input Filter

The Input Filter module performs three main functions:

- Formats data from the SPI-4 Rx FIFO into a 128-bit data bus with flags.
- Discards frames <= 16 bytes.
- Detects illegal SPI-4 burst sizes.

The Input Filter writes its output to the Burst FIFO.

2.1.3 Burst FIFO

The Burst FIFO is a 64x144 FIFO immediately upstream of the Input Controller. The Burst FIFO receives data from the Input Filter no faster than one qword every other cycle, but receives data continuously. The Input Controller, however, only accepts data from the Burst FIFO approximately half the time during the Reassembly Buffer "write window", but during the write window needs data every cycle to optimize Reassembly Buffer throughput. The Burst FIFO thus provides rate buffering between the Input Filter output and the Input Controller input.

2.1.4 Input Controller

The Input Controller reassembles frames from SPI-4 bursts, manages the Input Reassembly Buffer, generates SPI-4 receive FIFO status back to the SPI-4 Module for output to the SPI-4 transmitter, and forwards frames to the CRC Engine. Management of the Reassembly Buffer requires the Input Controller to track and store the state of up to 64 queues. These queue state records are stored in the Queue State RAM.

2.1.5 Reassembly Buffer

The Reassembly Buffer is a 34Kbyte (2048x135 bits) block of single-port SRAM that can be flexibly partitioned into up to 64 independent queues. The data path through the RAM is 128 bits wide with an additional 7 bits for flags. The Input Controller manages the Reassembly Buffer.

2.1.6 Queue State RAM

The Queue State RAM (64x100) holds configuration and state information for 64 queues and is used by the Input Controller to manage the Reassembly Buffer and track queue state.

2.1.7 CT Timeout RAM

The CT Timeout RAM (64x8) holds the cut-through timeout configuration values for 64 ports. The Input Controller uses the timeout values to determine when to declare that an input port is "hung". Refer to section 3.1.4.5 for details on the cut-through timer.

2.1.8 CRC Enable RAM

The CRC Enable RAM (8x32) holds the CRC Enable configuration bits for 256 ports. When high, the CRC Enable bit indicates that a port can carry CRC traffic. The Input Controller presents the CRC Enable bit to the CRC Engine with each input frame.

2.2 CRC Engine

The CRC Engine performs CRC processing on incoming frames in accordance with the Host Message API HLD. The CRC header defined in that document must be present as the first qword of all frames that require CRC processing. The CRC Engine also detects CRC payload length errors (end of frame arrives before end of CRC payload), and missing frame end flags. In both cases the CRC Engine sets the frame error flag and asserts an interrupt.

2.3 Frame Forwarder

The Frame Forwarder performs the following functions:

- Forwards data from the CRC Engine to the Scratchpad (complete frames) and Input Processing Module (frame headers).
- Detects missing frame start flags and re-frames data as necessary to ensure that only structurally valid frames are forwarded. Sets frame error flag and asserts an interrupt when an error is detected.
- Forwards headers to the Logical Processors in the IPM in a fixed round-robin order.
- Presents side information to the Logical Processor with each frame header.
- Meters data to the Scratchpad at the maximum allowed Scratchpad rate (1 qword every 3 cycles).
- Monitors Scratchpad and IPM backpressure.
- Waits for the Scratchpad to issue a frame ID for each frame before forwarding the next frame.

The Frame Forwarder contains the Port Information Table (a 256x15 RAM) to hold the VLAN_ID and PP_CODE_OFFSET configuration values for 256 ports. These values are part of the side information presented to the IPM at the start of each frame.

2.4 Input Processing Module

Figure 3 is a block diagram of the Input Processing Module (IPM). The IPM processes frame headers and generates input events to the Dispatcher through the Input Event Queue. The following paragraphs provide a brief overview of the main IPM components. For more detailed information on module operation, please refer to section 3.2.

2.4.1 Packet Processor

The Packet Processor (PP) includes two identical Logical Processor (LPs), a control store memory, and registers. The two LPs (LP0 and LP1) share the control store memory and other PP resources, but are completely independent processors. Please refer to the PP HLD for details.

2.4.2 Input Header Buffer

The IPM contains two Input Header Buffers. Each Input Header Buffer receives and stores frame headers from the Frame Forwarder and provides a FIFO-like read interface to one of the two LPs in the Packet Processor. Each Input Header Buffer contains two 256-byte buffers in a ping-pong configuration to allow the LPs to process one buffer while the other is filling.

2.4.3 Release Command Queue

The Release Command Queue buffers one release command from each LP and presents the commands one at a time to the Scratchpad as allowed by a Scratchpad busy signal.

2.4.4 Input Event Queue

The Input Event Queue sits between the Packet Processor and the Dispatcher. It stores up to sixteen input events (eight from each LP) and bursts them to the Dispatcher one at a time upon request. The Input Event Queue consists of the Event Queue Controller and two Event Queue RAMs.

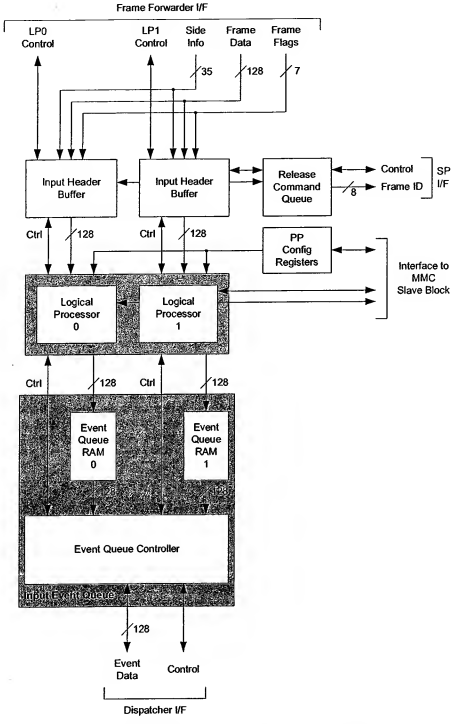

Figure 3: Input Processing Module Block Diagram

3 IPU Functional Operation

The following sections describe the details of IPU operation.

3.1 Input Receive Module

This section describes the architecture and operation of each Input Receive Module component.

3.1.1 SPI-4 Rx FIFO

The SPI-4 Module has no buffering in its receive path and thus can not be directly backpressured (backpressure is applied indirectly via FIFO status to the SPI-4 transmitter), so the IPU must accept data whenever the SPI-4 Module presents it. Also, as described in detail in reference [2], the SPI-4 module drives data on two paths (Main and Alt) on the same cycle in some situations. The Main data path is 64 bits wide and Alt is 32 bits. The SPI-4 Rx FIFO (64x124) buffers both data paths then merges them into a single 64-bit data bus with flags to the Input Filter. The SPI-4 Rx FIFO pushes data at the Input Filter and cannot be backpressured.

3.1.2 Input Filter

The Input Filter receives 64-bit data words plus flags from the SPI-4 Rx FIFO, formats them into 128-bit qwords plus flags, and writes them to the Burst FIFO. The Input Filter also filters out (discards) runt frames <= 16 bytes. An interrupt is generated and a runt frame counter increments for each runt frame discarded.

3.1.3 Burst FIFO

The Burst FIFO (64x144) buffers data and flags from the Input Filter and pushes data at the Input Controller when allowed by the Input Controller (the burst FIFO can be backpressured). The Burst FIFO inserts one dead cycle before each port number change to allow time for the Input Controller queue state to update before it is written back to the Queue State RAM.

3.1.4 Input Controller

The Input Controller performs the following functions:

- **Reassembly Buffer Management**: Allows flexible partitioning of the buffer into up to 64 independent queues and manages read and write accesses to the queues.
- **Frame Reassembly**: Moves data from the Burst FIFO to port queues in the Reassembly Buffer to assemble complete frames from SPI-4 bursts.
- **Frame Forwarding**: Forwards complete frames to the CRC Engine when scheduled via the Read Command Queue.
- **Frame Order Preservation**: Ensures that input frame order is maintained to the IPM and Scratchpad by scheduling frames for forwarding through the Read Command Queue based on order of arrival.
- **SPI-4 FIFO Status Generation**: Tracks queue status and generates FIFO status updates to the SPI-4 Module based on per-port queue levels.
- **Bad Frame Filtering**: Detects certain frame errors and either discards bad frames immediately or tags them as bad so they can be discarded downstream.
- **Full-Frame Mode**: Supports full-frame mode where N SPI-4 ports share a single Reassembly Buffer queue and complete frames must be transferred across the SPI-4 bus (vs. interleaved bursts).
- **Cut-Through Timer**: Operates the cut-through timer when a frame is forwarded in cut-through mode to detect a hung port condition that could block all IPU traffic.

The Input Controller maintains a Queue State RAM that holds state information for up to 64 queues numbered 0-63. Table 1 shows the state information for one queue. The Queue State RAM is organized as 64x100 with one line per queue so the entire 100 bits of state information can be accessed in a single cycle.

Field	Bits	Description
wr_cut_thru_state	1	'1' when queue in cut-thru mode.
wr_dump_til_eop_state	1	'1' when dumping data through next EOP.
port_valid	1	'1' when port is valid. (Configuration parameter.)
first_line	11	First line in queue. (Configuration parameter.)
last_line	11	Last line in queue. (Configuration parameter.)
wr_addr	11	Current write address. (Configuration parameter.)
rd_addr	11	Current read address. (Configuration parameter.)
queue_level	11	Number of qwords currently in queue.
aempty_level	11	Queue level corresponding to STARVING/HUNGRY transition. (Configuration parameter.)
afull_level	11	Queue level corresponding to HUNGRY/SATESFIED transition. (Configuration parameter.)
frm_count	10	Number of frames currently in queue. Used for error checking and to detect a blocking condition.
cur_frm_qw_cnt	10	Number of qwords in current frame. Used to detect runt and jabber frames.
Total	100	

Table 1: Queue State Definition

3.1.4.1 Queue Configuration

The Reassembly Buffer is partitioned at configuration time by setting the *first_line* and *last_line* parameters for each queue to be defined. A dedicated queue must be configured for each SPI-4 port (when not in full-frame mode), and the Queue State RAM entry number must match the associated SPI-4 port number. The only other restraint on partitioning the Reassembly Buffer is that queues must not overlap. A common configuration would be N equal-sized queues (N = the number of active SPI-4 ports), but the queue sizes need not be the same. Once the queue sizes are set the a*empty_level* and a*full_level* watermarks must be configured, and *wr_addr* and *rd_addr* must be set equal to *first_line*. The *port_valid* bit must be set to '1' for all active ports and should be set to '0' for all inactive ports to allow detection of invalid port addresses. All other Queue State RAM fields must be initialized to 0.

3.1.4.2 CRC Enable RAM

The CRC Enable RAM is an 8x32 (256 bits total) register array that contains the CRC Enable configuration bit for each SPI-4 port. The CRC Enable bit defines whether a port carries Host or non-Host traffic. Host traffic is defined as traffic that abides by the Pegasus Host Message API requirements. Each time the Input Controller forwards a frame it uses the frame's SPI-4 port number to fetch the associated CRC Enable bit from the CRC Enable RAM. The CRC Enable bit is then presented to the CRC Engine with the frame.

3.1.4.3 Reassembly Buffer Operation

The preferred mode of operation for the Reassembly Buffer is "store-and-forward", where a frame is not forwarded until it has been completely received. However, store-and-forward operation is not possible unless all port queues are large enough to hold the largest possible frame they may receive. Table 2 shows the maximum number of SPI-4 ports that allows complete store-and-forward operation for three different maximum frame lengths. The table assumes a 32Kbyte Reassembly Buffer and 50% headroom in each port queue (i.e., the almost full watermark is set at queue_size/2).

Max Frame Size	Total Queue Size Required	Max # Ports for Store and Forward
1522 bytes	3044 bytes	10
9600 bytes	19200 bytes	1
16256 bytes	32512 bytes	1

Table 2: Max Ports That Allow Store-and-Forward Operation

When a received frame is larger than ½ the target Reassembly Buffer port queue, the queue will hit its almost full watermark (and apply SPI-4 backpressure) before the end of the frame (EOP in SPI-4) arrives, so

the frame must be forwarded before it has been completely received. Otherwise, the queue could become permanently blocked waiting for the EOP with the sender unable to send it due to SPI-4 backpressure. This "cut-through" mode of operation is generally undesirable for the following reasons:

1. The amount of time frames spend in the Scratchpad is no longer directly under Pegasus control. It now depends on how long it takes to receive the EOP from the SPI-4 transmitter vs. how long it takes a protocol core to figure out what to do with it.
2. If for some reason the port sourcing a large frame hangs before the EOP is received, the entire SPI-4 interface will eventually be blocked waiting for an EOP on the hung port. The cut-through timer will detect and clear this condition, but results in data loss.
3. IPU throughput depends on the behavior of the SPI-4 transmitter (see section 3.6.1).

Figure 4 attempts to illustrate typical Reassembly Buffer operation. The figure shows a snapshot of what the Reassembly Buffer and Read Command Queue would look like after operating with four active SPI-4 ports if all queues started empty and no queues were serviced. All four port queues have filled beyond their almost-full watermarks (set at the same level in this example) so backpressure has been applied to the SPI-4 transmitter and input data flow has stopped.

Some things to note in Figure 4:

- Frames are removed from the Reassembly Buffer in order of EOP arrival, not start-of-frame (SOP in SPI-4) arrival. This is illustrated by the order of read commands in the Read Command Queue.
- The FIFO nature of the Reassembly Buffer port queues ensures that input frame order within each port is preserved.
- The Port 1 queue is above the almost full watermark, but the queue does not yet contain an EOP. Normally a read command is not generated until an EOP is received, but in this case the queue cannot accept any more data so the EOP will never arrive and the queue is blocked. This condition occurs when a frame arrives that exceeds the queue size (as discussed above). The Input Controller detects a blocking condition as (*queue_level* >= *afull_level)* & (*frm_count* == 0) in the queue state (see Table 1). At this point a read command must be generated to make room for the EOP to arrive (see the last entry in the Read Command Queue). When the EOP finally does arrive, the Input Controller knows not to generate another read command because the *wr_cut_thru_state* flag will be set.

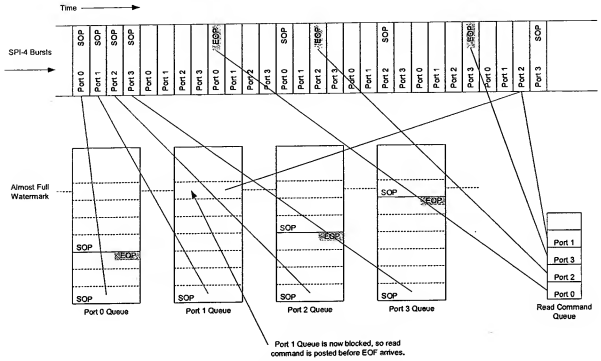

Figure 4: Reassembly Buffer Operation

3.1.4.4 SPI-4 FIFO Status Generation

The SPI-4 Phase 2 specification (reference [1]) defines three receiver FIFO states: STARVING (below almost empty watermark), HUNGRY (between almost empty and almost full watermarks), and SATISFIED (above almost full watermark). SPI-4 also defines two burst lengths, MAXBURST1 and MAXBURST2. The SPI-4 transmitter can burst up to MAXBURST1 16-byte blocks when the receiver FIFO status is STARVING, up to MAXBURST2 16-byte blocks when the FIFO status is HUNGRY, and can not start any new data transfers when the FIFO status is SATISFIED.

The Input Controller will continuously compare the *queue_level* of active queues against the configurable *aempty_level* and *afull_level* parameters. When a watermark is crossed a FIFO status update will be written to the SPI-4 Module, addressed by port number.

The SPI-4 Module defaults to SATISFIED on all ports following reset, so to avoid a startup deadlock condition the Input Controller asserts a global FIFO status update to the SPI-4 Module that sets all ports to HUNGRY. This occurs each time the Input Controller is enabled (ic_enable set to '1' in IRM_CONTROL – see 4.5.7.), and allows SPI-4 traffic flow to begin after startup. Thereafter, FIFO status updates are triggered only by port queue watermark crossings.

3.1.4.5 Cut-Through Timer

When the Input Controller starts forwarding a frame before the complete frame is present in the Reassembly Buffer (cut-through mode), the IPU can suffer complete deadlock if the associated SPI-4 port hangs before the end of the frame is received. This condition is not self-clearing, so a timer is used to detect it. The Input Controller starts the Cut-Through Timer as soon as cut-through mode is entered. Each port queue has an *Idle Timeout Value* parameter configured in the Cut-Through Timeout RAM that individually sets the timeout value for that port. Each time data arrives from the SPI-4 interface, the timer is cleared. If the timer reaches the timeout value between SPI-4 bursts, then the Input Controller forces an EOP (truncates the frame) with the error flag set and sets the *wr_dump_til_eop_state* flag in the queue state to recognize that any further

data on that port should be dropped up through the next EOP. The next EOP that arrives returns the queue state to normal. Note that if the EOP for the truncated frame never arrives, then the first "good" frame that arrives after the timer expiration will be dropped.

Forcing the EOP with error flag set allows the Input Controller to complete the frame read operation and clear the blocking condition. The error flag is passed to the Scratchpad with the last word of the frame and a protocol core will discard the truncated frame.

The Cut-Through Timer is 8 bits and has a configurable tick value that makes the total timer range variable. The tick value will support a maximum timeout range of approximately 60 seconds.

3.1.4.6 Frame Order Preservation and Fairness

Preservation of input frame order within each SPI-4 port is an ACP system requirement and is ensured under all conditions by the FIFO behavior of the port queues in the Reassembly Buffer.

Fairness in servicing SPI-4 ports is a system goal and is ensured under normal conditions by use of the Read Command Queue to schedule frame transfers out of the Reassembly Buffer. However, when frames are forwarded in cut-through mode fairness is not guaranteed under all conditions. As discussed in section 3.6.1, IPU throughput may suffer when cut-through occurs if the IPU and SPI-4 transmitter do not meet certain requirements. When IPU throughput suffers, fairness is also likely to suffer.

3.1.4.7 Error Handling

The Input Controller can detect the following errors (but not necessarily fix):

- Hung SPI-4 ports
- Runt frames
- Jabber frames
- SPI-4 frame aborts
- SPI-4 framing errors (missing SOP or EOP)
- Illegal SPI-4 burst sizes
- Illegal SPI-4 port address
- Port queue overflows

A hung SPI-4 port can be detected only when a frame is being forwarded in cut-through mode from that port. This was discussed in detail in section 3.1.4.5).

The Input Controller maintains a current frame qword count (*cur_frm_qw_cnt* in Table 1). This count is used to detect runt frames (frames that are illegally small) and jabber frames (frames that are illegally large).

When the Input Controller detects a runt frame (a frame <= *min_frm_size* in IRM_CONTROL), it writes the frame to the Reassembly Buffer normally but sets the Dump flag in the Read Command Queue along with the port number for the frame. The frame is then dropped by the Input Controller before it is forwarded.

When the Input Controller detects a jabber frame, it truncates the frame and sets the error bit with the last word in the Reassembly Buffer. It then discards the remainder of the frame as it arrives. The frame is forwarded normally and will be dropped downstream by a protocol core. Note that a jabber frame is defined as a frame that is > max_frm_size in IRM_CONTROL. This mechanism is intended to protect Pegasus from being flooded with garbage data from a malfunctioning port. Note that after a jabber frame is truncated, all traffic from that SPI-4 port is discarded up through the next frame end. If the end of the jabber frame never comes (in the form of a valid SPI-4 EOP flag), then the first "good" frame that arrives will be dropped.

The SPI-4 control word that terminates each frame can indicate that the preceding frame contained an error and should be aborted. This "frame abort" indication comes from the SPI-4 transmitter and will be passed to

the Scratchpad as the *frame_err* bit so a protocol core can drop the frame. The SPI-4 abort flag does not trigger an error interrupt by the IPU as it is considered a normal SPI-4 event.

The Input Controller can detect some framing errors (missing SOP or EOP). These errors are detected at the Reassembly Buffer output while frames are being forwarded. When detected, no corrective action is taken, but an interrupt is asserted. Note that framing errors are detected and fixed downstream of the Input Controller, so it may make sense to disable the Input Controller framing error interrupt to avoid redundant interrupts.

All SPI-4 bursts must be a multiple of 16 bytes long unless they end with an end-of-frame (EOP). The Input Filter detects illegal burst lengths and asserts an interrupt. The last 8 bytes of the illegal burst must be discarded when this occurs because only 16-byte qwords can be written to the IRM Reassembly Buffer. If this 8-byte fragment contains a start-of-frame (SOP), then the missing SOP will be detected in the Frame Forwarder and the frame error flag to the Scratchpad will be set. In all other cases, however, the IPU will not detect the missing data and the frame will not be flagged as containing an error. The system must then rely on software to catch the error.

The Input Controller uses two mechanisms to detect illegal port addresses. First, each entry in the Queue State RAM contains a *port_valid* bit that should be initialized to '1' for active ports and '0' for unused ports. A SPI-4 burst addressed to an unused port is invalid.. This mechanism is used when the IPU is not in full-frame mode, when the number of SPI-4 ports is limited to 64 and each port has a separate entry in the Queue State RAM. In full-frame mode, when a single port queue is used and all 256 SPI-4 ports can be addressed, the Input Controller compares the incoming port number to the "top_port_num" field in the IRM_CONTROL register. If the incoming port number is greater than top_port_num, the burst is invalid. In either case, the Input Controller discards all data addressed to an invalid port.

For a complete description of IPU frame error detection please refer to section 3.3.

3.1.4.8 Full-Frame Mode

The IPU can be configured to operate in a proprietary "full-frame" mode where N SPI-4 ports share a single receive queue in the Reassembly Buffer and a single FIFO status applies globally to all ports. This mode is outside the normal operation of the SPI-4 bus and must be explicitly supported by an interface device. In full-frame mode only complete frames can be transferred across the SPI-4 bus vs. the interleaved bursts normally used. The SPI-4 Module operates normally, but the FIFO status calendar must be configured with a single FIFO status (CALENDAR_LEN = 1) that applies globally to all ports. The Input Controller must be explicitly configured for full-frame mode by setting the full_frame_mode bit in IRM_CONTROL. In full-frame mode the Input Controller operates exactly as it does with a single SPI-4 port except that the port number is not used to address the Queue State RAM as it normally is.

3.1.4.9 Small-Burst Mode

The IPU can be configured to operate in a proprietary "small-burst" mode where STARVING SPI-4 FIFO status is only broadcast when a port is forwarding a frame in cut-through. Small-burst mode is intended to improve IPU throughput when the number of active SPI-4 ports is high. As the number of ports increases, the port queues in the IPU Reassembly Buffer become smaller, and the IPU spends more time in cut-through. As described in section 3.6.1.1, cut-through operation can significantly reduce IPU throughput when the number of ports is high.

Normally, a port queue is STARVING when below its almost-empty watermark, HUNGRY when between its almost-empty and almost full watermarks, and SATISFIED when above its almost-full watermark. In small-burst mode, the IPU broadcasts STARVING for a port only if the port queue is below almost empty AND the port is currently forwarding a frame in cut-through. Otherwise, HUNGRY is broadcast for all ports below almost full. STARVING is broadcast normally for ports above almost full. Small-burst mode attempts to take advantage of the fact that when the IPU spends a lot of time in cut-through, SPI-4 backpressure (status = SATISFIED) will be applied on most or all ports except for the one currently forwarding a frame in cut-through. This keeps the channel from the SPI-4 transmitter to the IPU Reassembly Buffer empty except for

traffic on the forwarding port. This places the SPI-4 bus in a situation where the channel is normally empty with the transmitter waiting for credits from the IPU. When the IPU grants a burst to the transmitter, the transmitter should immediately respond with that burst with minimal latency. This allows the transmitter to be configured with a larger burst size without danger of overflowing the IPU port queue.

Small-burst mode has not been simulated enough to fully characterize it, but significant increases in throughput have been achieved in the limited cases tested. Some question remains as to whether some combination of port queue watermark settings and SPI-4 burst sizes can achieve the same behavior without small-burst mode. Regardless of this uncertainty, small-burst mode is now an available IPU feature.

3.1.5 CRC Engine

The CRC Engine sits between the IRM and the Frame Forwarder and calculates CRCs on a per-frame basis in accordance with the Host Message API HLD. The CRC Engine contains small input and output FIFOs to isolate CRC Engine internal timing from the modules that interface with it. The CRC Engine can calculate CRC values over any two non-overlapping fields in each frame using either the iSCSI or FCIP polynomial. CRC calculation is controlled by a 128-bit CRC Header that must pre-pend each frame that requires CRC processing. (The CRC header is defined in the Host Message API HLD.) Frames without a CRC header are passed through the CRC Engine unmodified.

The CRC Engine detects the presence of the CRC Header by the state of a one-bit crc_flag in the first qword of each input frame. If the first qword is a CRC header the crc_flag = '1', otherwise crc_flag = '0'. The crc_flag is only defined for Host traffic that adheres to the Pegasus Host Message API. In non-Host traffic the crc_flag bit position in the first qword of each frame is unknown, so the CRC Engine must know not to check the bit and just pass non-Host traffic through. The CRC Engine distinguishes between Host and non-Host traffic via the CRC Enable flag from the IRM. If the CRC Enable flag = '1' (Host traffic), the CRC Engine checks the crc_flag bit in the first qword of the frame. If the CRC Enable flag = '0' (non-Host traffic), the CRC Engine does not check the crc_flag and passes the frame through.

When a CRC Header is present, the CRC Engine removes it, uses it to control CRC calculations, deposits its CRC results into the header, and appends the header to the end of the frame as a CRC trailer and forwards the frame to the Scratchpad.

If the CRC Engine hangs due to a bad payload pointer in the CRC Header, it times out, forwards the frame with the error flag set, and generates an error interrupt. In addition, if the CRC Engine detects a missing end-of-packet (EOP), it forces an EOP with the error flag set and generates an error interrupt.

3.1.6 Frame Forwarder

The Frame Forwarder performs the following functions:

- Reads data from the CRC Engine output FIFO.
- Forwards frames to the Scratchpad and IPM Logical Processors (LPs).
- Zeroes unused bytes in the last word of each frame.
- Forwards frame headers to LP0 and LP1 in fixed round-robin order.
- Sends an 8-bit frame ID (from Scratchpad), 8-bit SPI-4 port number, 3-bit PP Code Offset, 12-bit VLAN_ID, and 4-bit Event Sequence Number (ESN) to the LP with each frame header. These fields comprise the *PP Side Info*.
- Monitors backpressure from the Scratchpad and LPs, and stops transferring frame data if backpressure from either is asserted. (Note that LP backpressure only blocks header transfers. The rest of the frame can still be forwarded to the Scratchpad.)
- Meters the transfer of frame data to average no more than one 128-bit word every 3 system clock cycles to the Scratchpad and LPs. (This is a Scratchpad requirement.)
- Ensures that all frames forwarded are correctly framed with *frm_start* and *frm_end* flags.
- Sets the *frm_err* flag for all frames forwarded with known errors.
- Ensures no frames < 3 qwords are forwarded.

The PP Code Offset and VLAN_ID fields of the PP Side Info are stored in the 256x15 Port Information Table (PIT), which is loaded at configuration time. The Frame Forwarder addresses the PIT with the port address of each frame.

The Frame Forwarder maintains a 4-bit, modulus-16 Event Sequence Number counter that increments by one each frame. This 4-bit number is sent to the LP, which uses it to address into the Input Event Queue. This scheme ensures that input frame order is preserved to the Dispatcher (see section 3.2.4.3).

3.1.7 IPU Warm Reconfiguration

The IPU must be able to be halted and reconfigured without affecting the rest of the ACP. The required sequence is as follows, and will be controlled manually through IRM_CONTROL and IRM_STATUS register bits (see 4.5.7 and 4.5.8):

1. Halt all incoming SPI-4 traffic. (Accomplished external to the ACP.)
2. Wait for the Read Command Queue to empty (rcq_empty = '1'). This indicates that all complete frames have been forwarded from the Reassembly Buffer.
3. The Reassembly Buffer should also be empty (rb_empty = '1'). If not, then any partial frames in the Reassembly Buffer will be lost.
4. Place the SPI-4 Rx core in reset. (See the SPI-4 Module HLD.)
5. Disable the Input Controller (set ic_enable to '0').

After the completion of this sequence the IPU and SPI-4 Modules can be reconfigured. Reconfiguration is transparent to everything downstream of the IPU Reassembly Buffer except that no traffic will flow. Frame Forwarder and Event Queue Controller internal state are maintained so the LP round-robin servicing order will pick up where it left off when the IPU is brought back on line.

The procedure for bringing the IPU back on line after reconfiguration is as follows:

1. Remove the SPI-4 Rx core from reset. (See the SPI-4 Module HLD.)
2. Enable the Input Controller (set ic_enable to '1').
3. Resume incoming SPI-4 traffic. (Accomplished external to the ACP.)

During reconfiguration the IRM Queue State RAM must be completely re-initialized. Any partial frames present in the Reassembly Buffer (rb_empty = '0') will be lost. It is the responsibility of the management CPU to perform a controlled shut down and prevent data loss.

3.2 *Input Processing Module*

The Input Processing Module (IPM), shown in Figure 3, consists of two Input Header Buffers, a Packet Processor, the Input Event Queue, the Release Command Queue, and configuration logic. The following sections provide details on IPM operation.

Early revisions of this document referred to the combination of an Input Header Buffer and Logical Processor as an "Input Processor". This terminology was confusing and has been abandoned. Instead, the term "Logical Processor" was introduced to describe the two independent processing elements inside the Packet Processor. These two LPs share a single control store RAM and other PP resources, but are completely independent from a processing standpoint. Please refer to the Packet Processor HLD for details of the PP architecture.

3.2.1 Input Header Buffer

The Input Header Buffer sits upstream of the Packet Processor (PP) and includes the Header Buffer RAM and the Header Buffer Controller. The signal interface between the Frame Forwarder and Input Header Buffer is shown in Table 3. The signal interface between the Input Header Buffer and the Packet Processor

is shown in Table 4. The signal interface between the Input Header Buffer and Release Command Queue is shown in Table 5.

Signal	Direction	Description
frm_data[127:0]	From FF	Frame data bus. Only valid when *frm_data_valid* asserted.
frm_start	From FF	High during first qword of each frame. Only valid when *frm_data_valid* asserted.
frm_end	From FF	High during last qword of each frame. Only valid when *frm_data_valid* asserted.
frm_err	From FF	Valid only when *frm_end* is high. *frm_err* == 1 indicates that the frame just completed contains is known to contain an error.
frm_size[3:0]	From FF	Number of valid bytes in last qword of frame (1-15 = 1-15 bytes valid, 0 = all bytes valid). Only valid when *frm_data_valid* asserted.
frm_data_valid	From FF	Frame data valid.. *lp_sel* must also be high to accept frame data.
lp_sel	From FF	High indicates current frame intended for this LP.
lp_ready	To FF	High when Input Header Buffer can accept a new header.
spi4_port_num[7:0]	From FF	SPI-4 port number for current frame. Only valid when *side_info_valid* asserted.
frame_id[7:0]	From FF	Frame ID for current frame. Only valid when *side_info_valid* asserted.
vlan_id[11:0]	From FF	VLAN ID for current frame (associated with port number). Only valid when *side_info_valid* asserted.
code_entry_point[2:0]	From FF	PP code entry point for current frame (associated with port number). Only valid when *side_info_valid* asserted.
event_seq_num[3:0]	From FF	Event sequence number for current frame. Only valid when *side_info_valid* asserted.
side_info_valid	From FF	1-cycle active-high strobe indicates side information valid.

Table 3: Input Header Buffer/Frame Forwarder Interface Signals

Signal	Direction	Description
hdr_data[127:0]	To PP	Header FIFO read port.
hdr_available	To PP	High when a complete header is available to be processed, the first word of the header is valid on *hdr_data*, and all side information is valid. Goes low after first FIFO read.
frame_id[7:0]	To PP	Frame ID of current frame.
port_num[7:0]	To PP	Destination SPI-4 port number for current frame.
event_num[3:0]	To PP	Event Sequence Number for current frame.
code_entry_point[2:0]	To PP	Configuration item passed from IRM. Selects port-specific code offset for PP.
byte_cnt[7:0]	To PP	Received byte count of current header. 0 = 256 bytes.
force_sp_data	To PP	If '1', PP hardware will force the SP_DATA bit in the event for the current frame to the value of *sp_data_val*, and if the PP issues a release for this frame it will be ignored. The Input Header Buffer sets force_sp_data to '1' for: - All frames > 256 bytes. - Frames <= 256 with *frm_err* == '1'.
sp_data_val	To PP	If *force_sp_data* == '1', the PP will set SP_DATA = *sp_data_val* in the event for the current frame. Currently, *sp_data_val* will always be set high when *force_sp_data* is high.
hdr_fifo_rd	From PP	Advances header FIFO read pointer. Current word should be latched before pointer is advanced (i.e., the first word in the FIFO is valid
pp_done	From PP	Asserted when PP has finished processing current header.
drop_frame	From PP	Valid only when *pp_done* is high. If high a release command will be issued to the Scratchpad for the frame just processed.

Table 4: Input Header Buffer/Packet Processor Interface Signals

Signal	Direction	Description
frame_id[7.0]	To RCQ	Frame_ID for frame to be released.
frame_id_valid	To RCQ	High for 1 cycle to latch release command.
rcq_ready	From RCQ	High when RCQ can accept a release command. A release command must not be submitted when *rcq_ready* is low.

Table 5: Input Header Buffer/Release Command Queue Interface Signals

3.2.1.1 Header Buffer RAM

The Header Buffer RAM is a 32x128 single-port RAM that stores two 256-byte headers in a ping-pong configuration. The ping-pong configuration allows the PP Logical Processor (LP) to operate on one buffer (the active buffer) while the other is being filled, which avoids having to stall the Processor while receiving a header from the Frame Forwarder. Header Buffer writes and reads are interleaved, and the Header Buffer Controller controls the RAM. The Header Buffer appears as a single 16x128 FIFO to the LP.

3.2.1.2 Header Buffer Controller

The Header Buffer Controller performs the following functions:

- Manages the Header Buffer RAM.
- Generates backpressure (*lp_ready*) to the Frame Forwarder when both buffers are full.
- Presents a 128-bit wide FIFO data interface to the Packet Processor.
- Switches the active buffer under control of the Packet Processor.
- Latches PP side info from the Frame Forwarder and presents it to the Packet Processor.
- Issues release commands to the Release Command Queue when indicated by the Packet Processor.PP release commands are only accepted for frames <= 256 bytes AND frm_err == '0'. In all other cases they are ignored and the SP_DATA bit in the event is forced to '1' (*force_sp_data* = '1' and *sp_data_val* = '1').

3.2.2 Packet Processor

Please refer to the Packet Processor High Level Design document.

3.2.3 Release Command Queue

The Release Command Queue latches and serializes one release command from each LP and presents the commands to the Scratchpad one at a time as allowed by the Scratchpad *sp_rls_busy* signal. The Scratchpad asserts the *sp_rls_busy* signal when it is unable to accept release commands due to internal resource contention. If an LP tries to assert a second release command before a prior command has been accepted by the Scratchpad, the second command will be held off (*rcq_ready* held low) and the Processor will be stalled until the command is accepted. Given the minimum frame spacing and the round-robin distribution of frame headers to the LPs it is very unlikely that a Processor will be stalled by release command latency.

3.2.4 Input Event Queue

Figure 5 shows a block diagram of the Input Event Queue. The Input Event Queue performs the following functions:

- Stores up to 16 256-byte events from the Packet Processor.
- Sends events to the Dispatcher upon request in a two-burst sequence. The first burst consists of the first two quad words of the frame and the second burst consists of the remainder of the frame. The second burst can be repeated if requested by the Dispatcher.
- Uses the event size field from the event to determine how much data to burst.
- Allows events to be written in any order from the two LPs but reads events in the fixed order established by the 4-bit Event Sequence Number assigned by the Input Receive Module.

The following sections describe the architecture and operation of the Input Event Queue in detail.

3.2.4.1 Event Queue RAM

The Event Queue is comprised of two separate 128x128 two-port RAMs, one for each LP, and each RAM holds eight 256-byte events. A two-port RAM has separate read and write ports (with separate address buses) that can be accessed simultaneously. The write side of each RAM is controlled directly by its associated LP and the RAMs have word enables to allow the LP to write 32 bits at a time.

The RAM read ports are controlled by the Event Queue Controller and are accessed 128-bits at a time. A 2:1 Read Data Mux selects which RAM sources data to the Dispatcher.

3.2.4.2 Event Queue Controller

The Event Queue Controller performs the following functions:

- Maintains a 16-bit Buffer Status register to track the valid (completely written) status of each frame buffer in the Event Queue RAMs.
- Enforces event processing order by reading from the Event Queue RAMs in strict sequential order corresponding to the Event Sequence Number. A frame must be marked as valid in the Buffer Status register before it can be read.
- Generates Event Queue RAM read addresses and controls the Read Data Mux.
- Generates *eq_ready* signals to the LPs to indicate availability of frame buffers.
- Extracts the length field from the second word of the event to determine how much data to burst to the Dispatcher.
- Sends events to the Dispatcher upon request in a two-burst sequence. The first burst consists of the first two quad words of the frame and the second burst consists of the remainder of the frame. The second burst can be repeated if requested by the Dispatcher.

3.2.4.3 Event Queue Operation

The Frame Forwarder assigns a 4-bit Event Sequence Number (ESN) to each header it forwards to the Packet Processor. The ESN increments with each header and indicates the order in which the corresponding events must be processed by the Dispatcher.

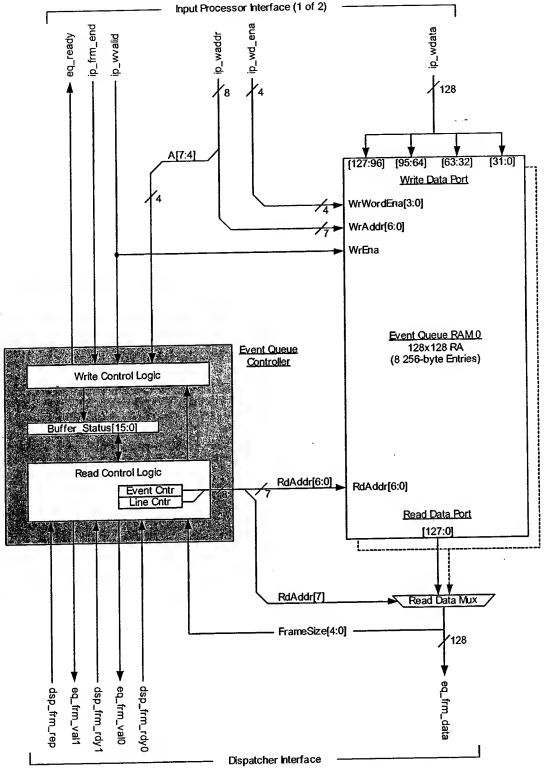

Figure 5: Input Event Queue Block Diagram

3.2.4.3.1 Event Queue Writes

The signal interface between a single LP and the Event Queue is shown in Table 6.

Signal	Direction	Description
lp_wdata[127:0]	From LP	Logical Processor write data bus. Valid when *lp_wvalid* is high.
lp_waddr[5:0]	From LP	Logical Processor write address bus (128-bit addressing). Valid when *lp_wvalid* is high.
lp_wd_ena[3:0]	From LP	Logical Processor word enables. Valid when *lp_wvalid* is high.
lp_wvalid	From LP	Logical Processor write valid signal. High indicates valid write cycle.
lp_frm_end	From LP	Frame end signal. High indicates last 32-bit write of an event.
eq_ready	To LP	Event Queue ready. High indicates a frame buffer is available.

Table 6: Logical Processor/Input Event Queue Interface Signals

Before starting to write an event to the Event Queue, the LP must verify that *eq_ready* is high. This signal indicates that the next frame buffer for this Processor is available. *eq_ready* will go low after the first write of a new frame and will stay low until after the frame is completed.

The LP indicates to the hardware when it is finished writing a frame by asserting the *lp_frm_end* signal high during the last write of the frame. After this last write the bit in the Buffer Status register (in the Event Queue Controller) corresponding to the Event Sequence Number for the frame just finished is set to '1'. This bit indicates to the Event Queue Controller read logic that the frame is valid (see section 3.2.4.3.2).

When an LP finishes writing an event the Event Queue Controller will check the bit in the Buffer Status register corresponding to the next frame buffer for that Processor. If the bit is clear (the Event Queue read process clears these bits) the buffer is available and *eq_ready* will be set high. If the status bit is still high then the Dispatcher has not yet received the event in that location and the LP must wait for the buffer to free up.

Events will generally be written to the Event Queue in the order they were delivered to the LPs, but variations in header processing time may prevent the exact input order from being preserved. The exact time order in which frames are written is not important, however, because the event read order is strictly enforced as described in the next section.

Section 3.2.4.4 lists all of the Event Queue access rules that must be adhered to by LPs.

3.2.4.3.2 Event Queue Reads

The signal interface between the Event Queue and Dispatcher is shown in Table 12.

While frames may be written to the Event Queue in any order, frames are always read in order of Event Sequence Number. The Event Queue Controller maintains a 4-bit Event Counter that tracks Event Sequence Numbers (i.e., it counts modulus 16 and increments by one for each event sent to the Dispatcher). This counter selects which of the 16 frame buffers in Event Queue RAM is next to be read, and points to the corresponding bit the Buffer Status register to determine when the frame can be read. Figure 6 shows where frame buffers 0-15 are located in Event Queue RAM.

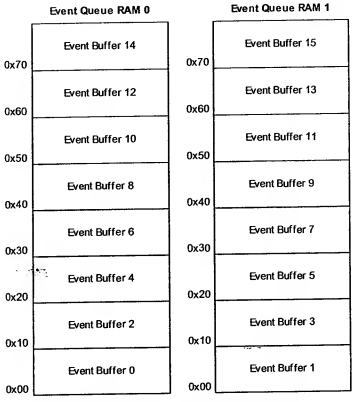

Figure 6: Event Queue Frame Buffer Numbering

The Event Queue Controller also maintains a 4-bit Line Counter that is used to address the 16 128-bit lines within each frame buffer. The combined 8 bits of the Event Counter and Line Counter are used to form the physical RAM addresses and the Read Data Mux select lines as shown in Table 7.

Counter Bits	Read Address Bits	Function
Event_Cnt[3:1]	RdAddr[6:4]	Selects one of eight frame buffers within a single RAM. Common to all RAMs.
Event_Cnt[0]	RdAddr[7]	Selects one of two physical RAMs through Read Data Mux.
Line_Cnt[3:0]	RdAddr[3:0]	Selects lines within a single frame buffer. Common to all RAMs.

Table 7: Event Queue RAM Read Address Generation

At start of day the Event Counter, the Line Counter, and the Buffer Status register are all set to 0. RdAddr points at the first line in frame buffer 0, the Read Data Mux selects RAM 0, and the Event Counter points at Buffer Status bit 0. The first event sent to the Dispatcher after startup must be the first frame from LP0, which carries Event Sequence Number 0 and is stored in frame buffer 0. The Event Queue Controller read logic monitors Buffer Status bit 0 until is goes high indicating frame buffer 0 contains a valid event.

The event transfer takes place in two parts. First, the Dispatcher asserts *disp_frm_rdy0* to request the first two quad words of the event. Event Queue provides first two quad words and *frm_disp_val0* when corresponding buffer is filled with entire event. These two words provide the Dispatcher with the information it needs to perform an FDC lookup and request access to the Message Bus. When Message Bus access has been granted the Dispatcher asserts *disp_frm_rdy1* a second time to request the remainder of the event. Event Queue then provides the remainder of the event with *frm_disp_val1* asserted. Both bursts to the Dispatcher are contiguous, *frm_disp_val0, and frm_disp_val1* is high throughout the first burst and second

burst separately, and the second burst must start two cycles after *disp_frm_rdy1* is asserted. The event size in the second quad word determines the burst size. Note that minimum event size is 3 quad words. Normally when the frame transfer is complete the corresponding bit in the Buffer Status register is cleared, the 4-bit Event Counter is incremented by one, and the Line Counter is cleared. However, if the Dispatcher asserts *disp_frm_rpt_req* along with *disp_frm_rdq* when requesting the second burst of a frame, then the Event Queue Controller must be able to repeat the second burst. In this case the read state must be restored to what it was at the start of the previous burst and the burst will be repeated the next time *disp_frm_rdy1* is asserted. The burst can be repeated as many times as required by asserting *disp_frm_rpt_req* with *disp_frm_rdy1*, but at some point the event must be completed normally by asserting *dsp_frm_rdy1* with *dsp_frm_rpt_req* low before the Event Queue Controller move on to the next event.

3.2.4.4 Packet Processor Event Queue Access Rules

The LPs must adhere to the following rules when accessing the Event Queue:

- The LP must not start writing an event unless the *eq_ready* signal from the Event Queue Controller is high. *eq_ready* will go low immediately after the first write of the event and will not go high again until the current event has been completely written.
- The *lp_wvalid* bit from the LP acts as a write-enable and must be asserted high during each write cycle. *lp_waddr*, *lp_wdata*, *lp_wd_ena*, and *lp_frm_end* are only valid when *lp_wvalid* is high.
- The LP can write continuously to any RAM location during an event write. Between events, access must be granted via *eq_ready* before the next event write can start.
- The 3 MSBs of the Event Sequence Number must form address bits [6:4] during all accesses for a given event.
- LP address bits [3:0] select the line being written within each event buffer and the *lp_wd_ena[3:0]* lines select 32-bit words within each line.
- The *lp_frm_end* signal from the LP must be high during the last write of an event and low during all other writes.

3.3 IPU Error Handling

Table 8 lists error events that the IPU hardware can detect.

Event	Where Detected	Detection Mechanism	Action
Runt Frame	IRM	cur_frm_qw_cnt <= MinFrameSize	Drop prior to forwarding, assert interrupt, advance runt_frm_cnt and dropped_qw_cnt.
Jabber Frame	IRM	cur_frm_qw_cnt > MaxFrameSize	Truncate frame, forward with error flag set, assert int, adv jabber_frm_cnt and dropped_qw_cnt.
SPI-4 Abort	IRM	SPI-4 frame abort in control word	Pass through as error flag.
SPI-4 burst error	IRM	Burst size not multiple of 16 bytes and does not contain EOP.	Assert interrupt and discard last 8 bytes of illegal burst.
Missing SOP	IRM	SOP flag not set in first word after EOP	Assert interrupt.
Missing EOP	IRM	Unexpected SOP	Assert interrupt.
Invalid SPI-4 Port Address	IRM	Port marked as invalid in queue state RAM or port number > top_port_num.	Discard burst, assert interrupt, advance dropped_qw_cnt.
Cut-through timeout	IRM	Timer expires waiting for data on port in cut-through mode.	Force end-of-frame, forward with error flag set, assert interrupt.
Port Queue overflow	IRM	A write is attempted to a full port queue.	Block write (data lost) and assert interrupt.
CRC payload error	CRC Engine	End of frame arrives before end of CRC payload. CRC Engine times out waiting for end of payload.	Assert interrupt, append CRC trailer, and forward frame with error flag set.
Missing EOP	CRC Engine	SOP for next frame detected before EOP for current frame.	Assert interrupt, pad frame with extra qword, force EOP with error flag.

Event	Where Detected	Detection Mechanism	Action
Missing SOP	Frame Forwarder	SOP flag not set in first word after EOP.	Assert interrupt, force SOP, assert end-of-frame error flag, and forward frame.
Runt Frame	Frame Forwarder	Checks that all forwarded frames are >= 3 qwords.	If runt frame detected, asserts interrupt, extends frame to 3 qwords, asserts end-of-frame error flag, and forwards frame.

Table 8: IPU Error Events and Actions

3.4 IPU Backpressure Mechanisms

Frame data and frame headers travel different paths through the IPU and each path has backpressure mechanisms that ensure a graceful response to ACP internal resource contention.

3.4.1 Frame Data Path Backpressure Mechanisms

Frame data flows through the IPU as follows:

SPI-4 -> Input Receive Module -> CRC Engine -> Frame Forwarder -> Scratchpad

The frame data backpressure mechanisms are as follows:

- The Scratchpad provides a full flag to the Frame Forwarder when it reaches a programmable watermark of available buffers.
- When the Frame Forwarder sees backpressure from the Scratchpad it stops reading from the CRC Engine output FIFO.
- When the CRC Engine output FIFO reaches almost full, the CRC Engine stops reading from its input FIFO.
- When the CRC Engine input FIFO reaches almost full, the Input Controller in the IRM stops moving data from the Reassembly Buffer to the CRC Engine.
- When a port queue in the Reassembly Buffer rises above its programmed watermarks the Input Controller updates SPI-4 FIFO status for the port to apply backpressure to the SPI-4 transmitter.

3.4.2 Frame Header Path Backpressure Mechanisms

Frame headers flow through the IPU as follows:

SPI-4 -> Input Receive Module -> CRC Engine -> Frame Forwarder -> Input Header Buffer -> Logical Processor -> Input Event Queue -> Dispatcher

The frame header backpressure mechanisms are as follows:

- When the Dispatcher stops accepting events from the Input Event Queue, the queue will fill up.
- When each Logical Processor's 8-entry Event Queue RAM fills the Processor will stall and stop processing headers from the Input Header Buffer.
- When the two 256-byte buffers in the Input Header Buffer fill, the Input Header Buffer asserts backpressure to the Frame Forwarder.
- When the Frame Forwarder sees backpressure from the Logical Processors it stops reading from the CRC Engine output FIFO. (Note: The Frame Forwarder only checks LP backpressure at the start of each frame, whereas Scratchpad backpressure is monitored continuously.)
- When the CRC Engine output FIFO reaches almost full, the CRC Engine stops reading from its input FIFO.
- When the CRC Engine input FIFO reaches almost full, the Input Controller in the IRM stops moving data from the Reassembly Buffer to the CRC Engine.

- When a port queue in the Reassembly Buffer rises above its programmed watermarks the Input Controller updates SPI-4 FIFO status for the port to apply backpressure to the SPI-4 transmitter.

3.5 Input Frame Order Preservation

Preservation of input frame order within each SPI-4 port is an ACP system requirement. Intra-port frame order is preserved through the Input Receive Module as described in section 3.1.4.6. Frame order is preserved through the Input Processing Module and Input Event Queue as described in section 3.2.4.3.2. The following is an overall summary of how frame order is preserved through the IPU:

- Port traffic passes through the Reassembly Buffer port queues in the order it arrives from the SPI-4 interface. The FIFO behavior of the port queues ensures that frame order within each SPI-4 port is maintained when the Input Controller reads frames from the Reassembly Buffer.
- The Frame Forwarder forwards frame headers to the LPs in a fixed round-robin fashion. It assigns a 4-bit Event Sequence Number to each header that increments by one with each header (Event Sequence Number counts modulus 16).
- The LPs write events into the Event Queue slot pointed to by the Event Sequence Number (the Event Queue has 16 slots numbered 0-15).
- Events are read from the Event Queue and sent to the Dispatcher in strict order of Event Sequence Number.

The Event Sequence Number ensures that the Dispatcher processes events in the same order that the associated data frames are forwarded from the Reassembly Buffer, and the FIFO behavior of the port queues in the Reassembly Buffer ensures that frames are forwarded from each port queue in the order they arrived on the associated SPI-4 port.

3.6 Expected Performance

3.6.1 Expected Input Receive Module Throughput

The IRM Reassembly Buffer is a single-port RAM, so read and write accesses cannot occur simultaneously. Therefore, support for 10Gbps throughput requires a combined read and write throughput of 20Gbps. The Reassembly Buffer RAM is 128 bits wide (not counting the 7 flag bits) and can be accessed every cycle, so its theoretical total throughput is 34Gbps (128 bits per cycle at 266MHz). Thus, RAM access must be 59% efficient to achieve the required 20Gbps performance.

Due to the overhead required to switch queue state in and out of the Queue State RAM, 59% efficiency will not be achieved under all conditions. On the input (write) side of the Buffer, small SPI-4 burst size and/or small frame size increase context-switching overhead. On the output (read) side of the Buffer, small frame size increases context-switching overhead. In addition, transitions between reading and writing incur several cycles of overhead, but this overhead is not affected by traffic characteristics. Exact throughput is difficult to estimate due to the dynamic nature of the IRM with normal input traffic.

Overhead due to queue state context-switching and read/write transitions is minimized in full-frame mode because the queue state is held in registers and never written back to the Queue State RAM.

3.6.1.1 Effect of Cut-Through on IRM Performance

Although store-and-forward is the desired mode of operation for the Reassembly Buffer for the reasons listed in section 3.1.4.3, store-and-forward mode does not always translate to better buffer performance (i.e., throughput). The Input Receive Module will always forward complete frames to the Scratchpad, regardless of whether the Reassembly Buffer is able to perform complete frame reassembly. By necessity, when a frame is being forwarded to the Scratchpad from one port queue no other ports queues are being serviced. Despite this apparent limitation, as long as the following conditions are met maximum Reassembly Buffer throughput can be maintained on all ports:

1. There must be enough buffering on all ports to hold one maximum-size frame. It does not matter where the buffering exists (it may be in the SPI-4 transmitter, the Reassembly Buffer, or a combination of both).
2. Frame transfers to the Scratchpad must occur at full 10Gbps line rate whether coming from the Reassembly Buffer in store-and-forward mode or coming from the SPI-4 transmitter and flowing through the Reassembly Buffer ("cut-through" mode).

When the Reassembly Buffer is in store-and-forward mode these requirements will always be met. In cut-through, however, we depend on the SPI-4 transmitter to help meet the requirements. The key is that when only one SPI-4 port is enabled the SPI-4 transmitter must be able to send data at the full 10Gbps SPI-4 data rate, not at the data rate of the individual port (10Gbps/N). This requires that the transmitter have a store-and-forward buffer, but it also requires that the transmitter receive transmit credits for the active port from the ACP at a rate of 10Gbps. (A SPI-4 transmitter can only send data as fast as it receives transmit credits from the receiver. Please refer to reference [1] for details on SPI-4 FIFO status operation.)

The problem is that with N SPI-4 ports (assume all equal bandwidth) the FIFO status for each port is only updated every N FIFO status clock cycles, which is equal to 8*N SPI-4 data path cycles (because the data path runs at 8x the FIFO status rate). The SPI-4 data path moves two bytes per cycle, so in order to issue credits at the rate of 10Gbps to one of N ports the burst size for that port must be 8*N*(2 bytes) = 16*N bytes. When N = 64 the required burst size is 16*64 = 1Kbytes.

In the N=64 case where the receive port queue for each port is 32Kbytes/64 = 512 bytes, a burst size of 1Kbytes is not possible. As the number of SPI-4 ports (N) increases the burst size required to maximize Reassembly Buffer throughput in cut-through mode increases linearly with N, while the port queue size in the Reassembly Buffer decreases at the rate of 1/N. With N ports the Reassembly Buffer size required to avoid cut-through is 4*N*(16*N) = 64*N^2. The factor of 4 accounts for the fact that the SPI-4 burst size will likely never be set larger than approximately ¼ of the port queue size. 64*N^2 = 32Kbytes at N = 22, which says that we should be in good shape for N <= 22.

It is important to note that the Reassembly Buffer is only forced into cut-through mode when an input frame exceeds the port queue size. (More accurately, cut-through occurs when an input frame exceeds the *afull_level* watermark, but that detail will be ignored during this discussion.) When the *maximum* frame size is smaller than the port queue size the Reassembly Buffer operates in store-and-forward mode and maximum throughput will always be achieved. In practice, average Reassembly Buffer throughput should be respectable with any number of ports as long as the *average* frame size is much smaller than the port queue size. The higher N is the smaller the average frame size must be for reasonable Reassembly Buffer throughput. With 64 ports the port queue size is 512 bytes, so throughput will be very poor for normal traffic (probably on the order of 2-4 times the throughput of a single port).

With 22 ports the port queue size about 1.5Kbytes, which allows a reasonable average frame size. When large frames do force the Reassembly Buffer into cut-through, 22 ports should allow maximum throughput as long as the SPI-4 transmitter has a store-and-forward buffer (see discussion earlier in this section). The qualitative conclusion of this analysis is that average Reassembly Buffer throughput should be good with 22 ports and should get increasingly better as N and/or the average frame size decrease(s). If the average frame size becomes too small, however, the overhead associated queue state context switching will begin to noticeably affect IRM throughput.

3.6.2 Expected CRC Engine and Frame Forwarder Throughput

Non-CRC traffic passes through the CRC Engine faster than the Frame Forwarder can forward it. The Frame Forwarder is hard-wired to forward traffic at a rate of one qword (128 bits) every three system clock cycles, which is the Scratchpad input limit. At 266MHz this data rate is 11.35Gbps.

CRC Engine throughput for CRC traffic is traffic-dependent for two reasons. First, internally the CRC Engine can feed data to the CRC generator either 8-bits or 64-bits at a time. Which data size is used depends on the CRC payload alignment. All 64-bit blocks that are 64-bit aligned within a qword are consumed in one cycle, but blocks that are not 64-bit aligned are consumed one byte per cycle. These non-aligned blocks can only

occur at the beginning and end of a CRC payload, so only very short payloads will suffer a noticeable throughput reduction.

The second throughput dependency for the CRC Engine involves where the last CRC payload ends in relation to the end of the frame. The CRC Engine suffers latency from the end of each payload until the CRC result is inserted in the CRC trailer. If the last CRC payload of the frame ends at the end of the frame, then this latency will delay forwarding the end of the frame. If the last CRC payload ends far enough before the end of the frame then the CRC result insertion latency will not affect throughput. The value of the latency is on the order of 6-8 cycles. Again, only short CRC frames will suffer a noticeable throughput reduction.

3.6.3 Expected Input Processing Module Throughput

In the absence of backpressure from the Dispatcher and Scratchpad, IPM throughput is limited only by the Packet Processor. Please refer to the Packet Processor HLD for PP throughput for various traffic types.

3.6.4 IPU Throughput Summary

Table 9 below lists IPU throughout observed during IPU/SPI-4 simulation (SPI-4 attached to IPU). Simulations are performed without backpressure from SP, DSP and assuming no latency on Packet Processor, while core clock runs at 266MHz, IRM reassembly buffer write window is 20 and read window is 24. SPI-4 module Tx stall FIFO water mark tx_stall_wm_high is 0x3C, and tx_stall_wm_low is 0x01.

Mode	SPI-4 Ports	Frmae size (byte)	CRC on/off	Burst Size Max1/Max2	SPI-4 Clock	Calendar Len	Thruput
full	256	48	off	64/32	350	Len=1, M=1	4.055
full	256	500	off	64/32	350	Len=1, M=1	6.755
full	256	16k	on	64/32	350	Len=1, M=1	6.291
full	256	48	off	512/256	350	Len=1, M=1	5.886
full	256	500	on	512/256	350	Len=1, M=1	10.437
full	256	16k	on	512/256	350	Len=1, M=1	9.708
Normal	64	48	off	64/32	350	Len=64, M=1	3.836
Normal	64	500	on	64/32	350	Len=64, M=1	1.314
Normal	64	16k	on	64/32	350	Len=64, M=1	0.63
Normal	64	48	off	128/64	350	Len=64, M=1	3.831
Normal	64	500	on	128/64	350	Len=64, M=1	2.508
Normal	64	16k	on	128/64	350	Len=64, M=1	1.268
Normal	8	48	off	64/32	350	Len=8, M=1	3.823
Normal	8	500	on	64/32	350	Len=8, M=1	9.556
Normal	8	16k	on	64/32	350	Len=1, M=1	4.584
Normal	8	48	off	512/256	350	Len=8, M=1	5.573
Normal	8	500	on	512/256	350	Len=8, M=1	10.543
Normal	8	16k	on	512/256	350	Len=8, M=1	10.002

Table 9: IPU Throughput Summary

4 Interfaces

4.1 Input Processing Unit/SPI-4 Core Interface

Signal	Direction	Description
main_data_valid	From SPI	Main Rx port data valid.
main_port_addr[7:0]	From SPI	Main Rx SPI-4 port address. Valid only when *main_data_valid* is high.
main_data[63:0]	From SPI	Main Rx port data. Most significant byte first. Valid only when *main_data_valid* is high.
main_sop	From SPI	Main Rx port start-of-packet. Valid only when *main_data_valid* is high.
main_eops[1:0]	From SPI	Main Rx port end-of-packet/abort: "00" = no EOP; "01" = EOP with abort; "10" or "11" = normal EOP (abort). Valid only when *main_data_valid* is high.
main_last_byte[2:0]	From SPI	Number of valid bytes in last word of packet: 1-7 = 1-7 bytes valid; 0 = all bytes valid. Valid only when *main_data_valid* is high.
alt_data_valid	From SPI	Alt Rx port data valid.
alt_port_addr[7:0]	From SPI	Alt Rx SPI-4 port address. Valid only when *alt_data_valid* is high.
alt_data[31:0]	From SPI	Alt Rx port data. Most significant byte first. Valid only when *alt_data_valid* is high.
alt_sop	From SPI	Alt Rx port start-of-packet. Valid only when *alt_data_valid* is high.
alt_eops[1:0]	From SPI	Alt Rx port end-of-packet/abort: "00" = no EOP; "01" = EOP with abort; "10" or "11" = normal EOP (abort). Valid only when *alt_data_valid* is high.
alt_last_byte[1:0]	From SPI	Number of valid bytes in last word of packet: 1-3 = 1-3 bytes valid; 0 = all bytes valid. Valid only when *alt_data_valid* is high.
status_valid	To SPI	Rx status data valid.
status_addr[7:0]	To SPI	Rx status SPI-4 port address. Valid only when *status_valid* is high.
status[1:0]	To SPI	Rx status: "00" = STARVING; "01" = HUNGRY; "10" = SATESFIED; "11" = invalid. Valid only when *status_valid* is high.
global_status_wr	To SPI	Valid only when *status_valid* is high. Updates status for all ports to value on *status[1:0]*.

Table 10: Input Processing Unit/SPI-4 Core Interface Signals

4.2 Input Processing Unit/Scratchpad Interface

Signal	Direction	Description
frame_data[127:0]	To SP	Frame data bus.
frame_data_valid	To SP	Asserted high to qualify *frame_data*, *frame_start*, *frame_end*, *frame_err*, and *byte_cnt*.
frame_start	To SP	High during first qword of frame.
frame_end	To SP	High during last qword of frame.
frame_err	To SP	Valid only during last qword of frame. High if frame contains an error.
byte_cnt[3:0]	To SP	Valid only during last qword of frame. Indicates number of valid bytes in last qword starting at MSByte ([127:120]). 1 = first (MS) byte valid, 2 = first 2 bytes valid, ..., 15 = first 15 bytes valid, 0 = all bytes valid. Invalid bytes in the last word must be set to zero.
frame_id[7:0]	From SP	Frame ID assigned by SP.
frame_id_valid	From SP	Asserted high when *frame_id* is valid. Asserted within 8 cycles after *frame_start*.
sp_full	From SP	High indicates SP has reached its configurable watermark of free buffers.
pkt_release	To SP	Active-high strobe tells SP to release buffers assigned to Frame ID *pkt_fid*.
pkt_fid[7:0]	To SP	Frame ID of frame to be release. Valid when *pkt_release* strobe is high.
sp_rls_busy	From SP	When high the Scratchpad cannot accept new release commands.

Table 11: Input Processing Unit/Scratchpad Interface Signals

4.3 Input Processing Unit/Dispatcher Interface

Signal	Direction	Description
eq_frm_data[127:0]	To Disp	Event Queue output data bus.
dsp_frm_rdy0	From Disp	Dispatcher ready for event burst 0 (first two quad words of event).
eq_frm_val0	To Disp	Data valid 0. High during event burst 0.
dsp_frm_rdy1	From Disp	Dispatcher ready for event burst 1 (remainder of event).
ed_frm_val1	To Disp	Data valid 0. High during event burst 1.
dsp_frm_rep	From Disp	Frame repeat. If high when dsp_frm_rdy1 goes high, event burst 1 will be repeated will be repeated the next time dsp_frm_rdy1 is asserted.

Table 12: Input Processing Unit/Dispatcher Interface Signals

4.4 Management Bus Interface

Please refer to the MMC HLD for interface details.

4.5 Configuration Registers

4.5.1 Register Map

Mode key: R/W = read/write, R = read only, W = write only, RTC = read to clear.

Offset	Mode	Register Name	Description
0000 – 0032	R/W	PP_EXT_CONFIG0-50	Packet Processor external configuration register
0033	R	IPM_STATUS	IPM status register
0034	R/W	IPM CONTROL	IPM control register
0035 – 0038	R/W	IPM_HDR_BUFR_DATA0-3	IPM header buffer W/R data through MMC
0039	R/W	IPM_HDR_BUFR_ADDR	IPM header buffer address through MMC

Offset	Mode	Register Name	Description
003a – 003d	R/W	IPM_EVNT_QUE_DATA0-3	IPM event queue W/R data through MMC
003e	R/W	IPM_EVNT_QU_ADDR	IPM event queue address through MMC
003f – 007f	n/a	reserved	reserved
0080-009F		PP internal register Please reference to PP HLD	
00a0 – 07ff	n/a	reserved	reserved
0800	R/W	IRM_CONTROL	IRM control register
0801	R	IRM_STATUS	IRM status register
0802	RTC	IRM_INT	IRM interrupt register
0803	R/W	IRM_INT_ENA	IRM interrupt enable register
0804	R/W	IRM_CT_TIMER_INC	IRM Cut-thru timer increment
0805	R/W	IRM_RB_TIMING_CTRL	IRM Reassembly Buffer R/W timing control
0806	R/W	IRM_QSR_ADDR	IRM QSR address
0807-080a	R/W	IRM_QSR_DATA0-3	IRM QSR data
080b	R/W	IRM_CTR_ADDR	IRM CTR address
080c	R/W	IRM_CTR_DATA	IRM CTR data
080d	R/W	IRM_CER_ADDR	IRM CER address
080e	R/W	IRM_CER_DATA	IRM CER data
080f	R/W	IRM_RB_ADDR	IRM reassembly buffer address
0810 – 0814	R/W	IRM_RB_DATA0-4	IRM reassembly buffer data
0815	RTC	IRM_RX_FRM_CNT	IRM received SPI-4 end-of-pkt count
0816-0817	RTC	IRM_RX_BYTE_CNT	IRM received SPI-4 byte count
0818	RTC	IRM_FWD_FRM_CNT	IRM forwarded end-of-pkt count
0819-081a	RTC	IRM_FWD_BYTE_CNT	IRM forwarded byte count
081b	RTC	IRM_RUNT_FRM_CNT	IRM runt frame counts
081c	RTC	IRM_JABR_CT_FRM_CNT	IRM jabber frame and cut-thru frame counts
081d	RTC	IRM_DROPD_QW_CNT	IRM dumped quad-word count
081e	R/W	IRM_RCQ_WRITE	IRM Read Command Queue write data
081f – 0fff	n/a	reserved	reserved
1000	R/W	FF_CONTROL	Frame Forwarder Control register
1001	R	FF_STATUS	Frame Forwarder Status register
1002	R/W	FF_PIT_ADDR	Frame Forwarder PIT RAM access address
1003	R/W	FF_PIT_DATA	Frame Forwarder PIT RAM access data
1004-17ff	n/a	reserved	reserved
1800	R/W	CRC_PYLD_ST_REF	CRC engine payload start reference
1801	R/W	IPU_DEBUG_MUX	IPU debug output mux select
1802	RTC	IPU_INT	IPU miscellaneous interrupts
1803	R/W	IPU_INT_ENA	IPU miscellaneous interrupt enables
1804-1fff	n/a	reserved	reserved

Table 13: IPU Register Map

4.5.2 PP External Configuration registers

4.5.2.1 PP_EXT_CONFIG0

Bits	Field Name	Description
31:0	exp_pmac_da[31:0]	32 LSB of Expected PMAC Destination Address

4.5.2.2 PP_EXT_CONFIG1

Bits	Field Name	Description
31:29	-	Not used
28	ipu_id	IPU ID
27:22	exp_tcp_flags	Expected value to be compared to LHB byte 13 [5:0] after masking
21:16	cf_mask	Create Flow Flag Mask
15:0	exp_pmac_da[47:32]	16 MSB of Expected PMAC Destination Address

4.5.2.3 PP_EXT_CONFIG2

Bits	Field Name	Description
31:0	pmac_mask[31:0]	32 LSB of Mask to be applied to DA before comparing to exp_pmac_da

4.5.2.4 PP_EXT_CONFIG3

Bits	Field Name	Description
31:16	-	Not used
15:0	pmac_mask[47:32]	15 MSB of Mask to be applied to DA before comparing to exp_pmac_da

4.5.2.5 PP_EXT_CONFIG4

Bits	Field Name	Description
31:0	exp_tmac_da[31:0]	32 LSB of Expected TMAC Destination Address

4.5.2.6 PP_EXT_CONFIG5

Bits	Field Name	Description
31:16	-	Not used
15:0	exp_tmac_da[47:32]	15 MSB Expected TMAC Destination Address

4.5.2.7 PP_EXT_CONFIG6

Bits	Field Name	Description
31:0	tmac_mask[31:0]	32 LSB of Mask to be applied to DA before comparing to exp_tmac_da

4.5.2.8 PP_EXT_CONFIG7

Bits	Field Name	Description
31:16	-	Not used
15:0	tmac_mask[47:32]	16 MSB of Mask to be applied to DA before comparing to exp_tmac_da

4.5.2.9 PP_EXT_CONFIG8

Bits	Field Name	Description
31:0	tcp_option_types[31:0]	31:0 of 10 possible TCP option types (for option parsing unit)

4.5.2.10 PP_EXT_CONFIG9

Bits	Field Name	Description
31:0	tcp_option_types[63:32]	63:32 of 10 possible TCP option types (for option parsing unit)

4.5.2.11 PP_EXT_CONFIG10

Bits	Field Name	Description
31:16	exp_protocol	Expected protocol value
15:0	tcp_option_types[79:64]	16 MSB of 10 possible TCP option types (for option parsing unit)

4.5.2.12 PP_EXT_CONFIG11

Bits	Field Name	Description
31:24	exp_l3a_len	Expected L3 address length
23:0	ip_option_types	3 possible IP option types (for option parsing unit)

4.5.2.13 PP_EXT_CONFIG12

Bits	Field Name	Description
31:0	flow_key_spdp_mask0	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b000

4.5.2.14 PP_EXT_CONFIG13

Bits	Field Name	Description
31:0	flow_key_spdp_mask1	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b001

4.5.2.15 PP_EXT_CONFIG14

Bits	Field Name	Description
31:0	flow_key_spdp_mask2	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b010

4.5.2.16 PP_EXT_CONFIG15

Bits	Field Name	Description
31:0	flow_key_spdp_mask3	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b011

4.5.2.17 PP_EXT_CONFIG16

Bits	Field Name	Description
31:0	flow_key_spdp_mask4	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b100

4.5.2.18 PP_EXT_CONFIG17

Bits	Field Name	Description
31:0	flow_key_spdp_mask5	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b101

4.5.2.19 PP_EXT_CONFIG18

Bits	Field Name	Description
31:0	flow_key_spdp_mask6	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b110

4.5.2.20 PP_EXT_CONFIG19

Bits	Field Name	Description
31:0	flow_key_spdp_mask7	Mask for LHB SP/DP field before writing into Event Structure for pp code offset= 3'b111

4.5.2.21 PP_EXT_CONFIG20

Bits	Field Name	Description
31:0	flow_key_sa_mask0	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b000

4.5.2.22 PP_EXT_CONFIG21

Bits	Field Name	Description
31:0	flow_key_sa_mask1	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b001

4.5.2.23 PP_EXT_CONFIG22

Bits	Field Name	Description
31:0	flow_key_sa_mask2	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b010

4.5.2.24 PP_EXT_CONFIG23

Bits	Field Name	Description
31:0	flow_key_sa_mask3	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b011

4.5.2.25 PP_EXT_CONFIG24

Bits	Field Name	Description
31:0	flow_key_sa_mask4	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b100

4.5.2.26 PP_EXT_CONFIG25

Bits	Field Name	Description
31:0	flow_key_sa_mask5	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b101

4.5.2.27 PP_EXT_CONFIG26

Bits	Field Name	Description
31:0	flow_key_sa_mask6	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b110

4.5.2.28 PP_EXT_CONFIG27

Bits	Field Name	Description
31:0	flow_key_sa_mask7	Mask for LHB SA field before writing into Event Structure for pp code offset= 3'b111

4.5.2.29 PP_EXT_CONFIG28

Bits	Field Name	Description
31:0	flow_key_da_mask0	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b000

4.5.2.30 PP_EXT_CONFIG29

Bits	Field Name	Description
31:0	flow_key_da_mask1	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b001

4.5.2.31 PP_EXT_CONFIG30

Bits	Field Name	Description
31:0	flow_key_da_mask2	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b010

4.5.2.32 PP_EXT_CONFIG31

Bits	Field Name	Description
31:0	flow_key_da_mask3	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b011

4.5.2.33 PP_EXT_CONFIG32

Bits	Field Name	Description
31:0	flow_key_da_mask4	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b100

4.5.2.34 PP_EXT_CONFIG33

Bits	Field Name	Description
31:0	flow_key_da_mask5	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b101

4.5.2.35 PP_EXT_CONFIG34

Bits	Field Name	Description
31:0	flow_key_da_mask6	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b110

4.5.2.36 PP_EXT_CONFIG35

Bits	Field Name	Description
31:0	flow_key_da_mask7	Mask for LHB DA field before writing into Event Structure for pp code offset= 3'b111

4.5.2.37 PP_EXT_CONFIG36

Bits	Field Name	Description
31:24	flow_key_ptcl_mask3	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b011
23:16	flow_key_ptcl_mask2	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b010
15:8	flow_key_ptcl_mask1	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b001
7:0	flow_key_ptcl_mask0	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b000

4.5.2.38 PP_EXT_CONFIG37

Bits	Field Name	Description
31:24	flow_key_ptcl_mask7	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b111
23:16	flow_key_ptcl_mask6	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b110
15:8	flow_key_ptcl_mask5	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b101
7:0	flow_key_ptcl_mask4	Mask for ptcl field before writing into Event Structure for pp code offset= 3'b100

4.5.2.39 PP_EXT_CONFIG38

Bits	Field Name	Description
31:24	flow_key_ptcl_force3	Force for ptcl field before writing into Event Structure for pp code offset= 3'b011
23:16	flow_key_ptcl_force2	Force for ptcl field before writing into Event Structure for pp code offset= 3'b010
15:8	flow_key_ptcl_force1	Force or ptcl field before writing into Event Structure for pp code offset= 3'b001
7:0	flow_key_ptcl_force0	Force for ptcl field before writing into Event Structure for pp code offset= 3'b000

4.5.2.40 PP_EXT_CONFIG39

Bits	Field Name	Description
31:24	flow_key_ptcl_force7	Force for ptcl field before writing into Event Structure for pp code offset= 3'b111
23:16	flow_key_ptcl_force6	Force for ptcl field before writing into Event Structure for pp code offset= 3'b110
15:8	flow_key_ptcl_force5	Force for ptcl field before writing into Event Structure for pp code offset= 3'b101
7:0	flow_key_ptcl_force4	Force for ptcl field before writing into Event Structure for pp code offset= 3'b100

4.5.2.41 PP_EXT_CONFIG40

Bits	Field Name	Description
31:16	constant_data1	Constant data value
15:0	Constant_data0	Constant data value

4.5.2.42 PP_EXT_CONFIG41

Bits	Field Name	Description
31:16	constant_data3	Constant data value
15:0	constant_data2	Constant data value

4.5.2.43 PP_EXT_CONFIG42

Bits	Field Name	Description
31:16	constant_data5	Constant data value
15:0	constant_data4	Constant data value

4.5.2.44 PP_EXT_CONFIG43

Bits	Field Name	Description
31:16	constant_data7	Constant data value
15:0	constant_data6	Constant data value

4.5.2.45 PP_EXT_CONFIG44

Bits	Field Name	Description
31:16	constant_data9	Constant data value
15:0	constant_data8	Constant data value

4.5.2.46 PP_EXT_CONFIG45

Bits	Field Name	Description
31:16	constant_data11	Constant data value
15:0	constant_data10	Constant data value

4.5.2.47 PP_EXT_CONFIG46

Bits	Field Name	Description
31:16	constant_data13	Constant data value
15:0	constant_data12	Constant data value

4.5.2.48 PP_EXT_CONFIG47

Bits	Field Name	Description
31:16	constant_mask0	Constant mask value
15:0	constant_data14	Constant data value

4.5.2.49 PP_EXT_CONFIG48

Bits	Field Name	Description
31:16	constant_mask2	Constant mask value
15:0	constant_mask1	Constant mask value

4.5.2.50 PP_EXT_CONFIG49

Bits	Field Name	Description

Bits	Field Name	Description
31:16	constant_mask4	Constant mask value
15:0	constant_mask3	Constant mask value

4.5.2.51 PP_EXT_CONFIG50

Bits	Field Name	Description
31:16	constant_mask6	Constant mask value
15:0	constant_mask5	Constant mask value

4.5.3 IPM Status Register (IPM_STATUS)

Bits	Field Name	Description
31:26	-	Not Used
25:10	event_status	Event status for event queue
9	event_queue_error	Event queue sequence number error
8	evtq_rd_idle	Event queue read in idle state
7	lp1_evtq_wr_idle	Event queue RAM1 write in idle state
6	lp0_evtq_wr_idle	Event queue RAM0 write in idle state
5	lp1_rcq_ready	LP1 release command is ready to accept new command
4	lp0_rcq_ready	LP0 release command is ready to accept new command
3	lp1_hdr_buffer_rd_idle	LP1 header buffer read in idle state
2	lp1_hdr_buffer_wr_idle	LP1 header buffer write in idle state
1	lp0_hdr_buffer_rd_idle	LP0 header buffer read in idle state
0	lp0_hdr_buffer_wr_idle	LP0 header buffer write in idle state

4.5.4 IPM Control Register (IPM_CONTROL)

Bits	Field Name	Description
31:3	-	Not used
2	lp1_reg_fifo_ena	High to enable LP1 header buffer
1	lp0_reg_fifo_ena	High to enable LP0 header buffer
0	reg_evt_ena	High to enable input event queue

4.5.5 IPM Header Buffer RAM Access

The Header Buffer RAM can only be accessed from the MMC when the Header Buffer is disabled. To disable Header Buffer0 in IP0, set ip0_reg_fifo_ena = "0" in IPM_CONTROL. To disable Header Buffer1 in IP1, set ip1_reg_fifo_ena = "0" in IPM_CONTROL. It is accessed as follows:

Writes:
1. Write HDR_BUFFER_DATA0-3 in order.
2. Write target Header Buffer address to HDR_BUFFER_ADDR. Set hdr_write High for RAM write.

Reads:
1. Write target Header Buffer address to HDR_BUFFER_ADDR. Set hdr_write Low for RAM read.
2. Read HDR_BUFFER_DATA0-3 in any order.

4.5.5.1 IPM Header Buffer data (HDR_BUFR_DATA0)

Bits	Field Name	Description
31:0	hdr_buffer_data0	[31:0] Data from Write/Read accessing Header buffer0/1

4.5.5.2 IPM Header Buffer data (HDR_BUFR_DATA1)

Bits	Field Name	Description

Bits	Field Name	Description
31:0	hdr_buffer_data1	[63:32] Data from Write/Read accessing Header buffer0/1

4.5.5.3 IPM Header Buffer data (HDR_BUFR_DATA2)

Bits	Field Name	Description
31:0	hdr_buffer_data2	[95:64] Data from Write/Read accessing Header buffer0/1

4.5.5.4 IPM Header Buffer data (HDR_BUFR_DATA3)

Bits	Field Name	Description
31:0	hdr_buffer_data3	[127:96] Data from Write/Read accessing Header buffer0/1

4.5.5.5 IPM Header Buffer address (HDR_BUFR_ADDR)

Bits	Field Name	Description
31:7	-	Not used
6	hdr_write	1: write access to header buffer 0; read access to header buffer
5	hdr_test_sel	1: select header buffer1; 0: select header buffer0
4:0	hdr_buffer_addr	Address for accessing Header Buffer0/1

4.5.6 IPM Event Queue RAM Access

The Event Queue RAM can only be accessed from the MMC when the Event Queue is disabled. To disable Event Queue, set reg_evtq_ena = "0" in IPM_CONTROL. It is accessed as follows:

Writes:
1. Write EVENT_QUEUE_DATA0-3 in order.
2. Write target Event Queue address to EVENT_QUEUE_ADDR. Set event_queue_write High for RAM write.

Reads:
1. Write target Header Buffer address to HDR_BUFFER_ADDR. Set event_queue_write Low for RAM read.
2. Read EVENT_QUEUE_DATA0-3 in any order.

4.5.6.1 IPM Event Queue data (EVNT_QUE_DATA0)

Bits	Field Name	Description
31:0	event_queue_data0	[31:0] Data for Write/Read accessing Header buffer0/1

4.5.6.2 IPM Event Queue data (EVNT_QUE_DATA1)

Bits	Field Name	Description
31:0	event_queue_data1	[63:32] Data for Write/Read accessing Header buffer0/1

4.5.6.3 IPM Event Queue data (EVNT_QUE_DATA2)

Bits	Field Name	Description
31:0	event_queue_data2	[95:64] Data for Write/Read accessing Header buffer0/1

4.5.6.4 IPM Event Queue data (EVNT_QUE_DATA3)

Bits	Field Name	Description
31:0	event_queue_data3	[127:96] Data for Write/Read accessing Header buffer0/1

4.5.6.5 IPM Event Queue address (EVNT_QUE_ADDR)

Bits	Field Name	Description
31:9	-	Not used
8	event_queue_write	1: write access to header buffer 0: read access to header buffer
7	event_queue_select	1: select event queue1 ; 0: select event queue0
6:0	event_queue_addr	Address for accessing Header Buffer0/1

4.5.7 IRM Control Register (IRM_CONTROL)

Bits	Field Name	Description
31:24	-	Not used
23:16	top_port_num	Sets top valid SPI-4 port number
15	bf_fifo_reset	'1' flushes burst FIFO
14	rcq_reset	'1' flushes receive command queue
13:7	max_frm_size	Maximum frame size, in 128-byte blocks. Frames larger than max_frm_size*8 + 1 qwords will be truncated.
6:3	min_frm_size	Minimum frame size, in 16-byte blocks. Frames <= min_frm_size qwords will be discarded. **NOTE: MINIMUM VALUE IS 2.**
2	-	Not used (must be '0').
1	full_frame_mode	'1' enables full frame mode
0	ic_enable	'1' enables IRM Input Controller

4.5.8 IRM Status Register (IRM_STATUS)

IRM_STATUS indicates "live" status. These bits are not sticky.

Bits	Field Name	Description
31:3	-	Not Used
2	bf_empty	Burst FIFO empty
1	rcq_empty	Release command queue empty
0	rb_empty	Reassembly buffer empty

4.5.9 IRM Interrupt Register (IRM_INT)

The IRM_INT register latches IRM error events and generates an interrupt if the corresponding bit in IRM_INT_ENA is high. The enable bits only affect interrupt generation. IRM_INT will always indicate latched error status. Reading IRM_INT clears all bits, but if an underlying error condition is still present, the corresponding bit will go high again immediately after the read.

Bits	Field Name	Description
31:10	-	Not Used
9	port_queue_overflow	'1' = one or more Reassembly Buffer port queues overflowed since last IRM_INT read
8	ct_timer_expired	'1' = cut-thru timer expired since last IRM_INT read
7	jabber_frame	'1' = jabber frame detected and truncated since last IRM_INT read
6	runt_frame	'1' = runt frame detected and discarded since last IRM_INT read
5	spi4_burst_err	'1' = illegal SPI-4 burst size detected since last IRM_INT read
4	framing_err	'1' = framing error (missing SOP or EOP) detected since last IRM_INT read
3	spi4_invalid_port	'1' = invalid SPI-4 port address received since last IRM_INT read
2	rcq_full	'1' = Read Command Queue overflow occurred since last IRM_INT read
1	sf_overflow	'1' = SPI-4 Rx FIFO overflowed since last IRM_INT read
0	bf_overflow	'1' = Burst FIFO overflowed since last IRM_INT read

4.5.10 IRM Interrupt Enable Register (IRM_INT_ENA)

Bits	Field Name	Description
31:10	-	Not Used
9	port_queue_of_ena	'1' = port queue overflow interrupt enabled
8	ct_timer_expired_ena	'1' = cut-thru timer expired interrupt enabled
7	jabber_frame_ena	'1' = jabber frame interrupt enabled
6	runt_frame_ena	'1' = runt frame interrupt enabled
5	spi4_burst_err_ena	'1' = illegal SPI-4 burst size interrupt enabled
4	framing_err_ena	'1' = framing error interrupt enabled
3	spi4_invalid_port_ena	'1' = invalid SPI-4 port address interrupt is enabled
2	rcq_full_ena	'1' = Read Comand Queue overflow interrupt is enabled
1	sf_overflow_ena	'1' = SPI-4 Rx FIFO overflow interrupt is enabled
0	bf_overflow_ena	'1' = Burst FIFO in overflow interrupt is enabled

4.5.11 IRM Cut Thru Timer Increment (IRM_CT_TIMER_INC)

Bits	Field Name	Description
31:26	-	Not Used
25:0	ct_timer_inc	IRM cut-thru timer increment in system clock ticks

4.5.12 IRM Reassembly Buffer Timing Control (IRM_RB_TIMING_CTRL)

Bits	Field Name	Description
31:29	-	Not used
28:24	min_wr_win_size	Sets minimum write window size when soft_win_ena = '1'.
23:19	max_wr_win_size	Sets hard wr win size when soft_win_ena = '0', and max wr win size when soft_win_ena = '1'.
18:16	wr2rd_gap	Sets gap between the write window and read window.
15:11	min_rd_win_size	Sets minimum read window size when soft_win_ena = '1'.
10:6	max_rd_win_size	Sets hard rd win size when soft_win_ena = '0', and max rd win size when soft_win_ena = '1'.
5:3	rd2wr_gap	Sets gap between the read window and write window.
2	wr_priority	'1' gives writes priority when soft_win_ena = '1' if rd_win has exceeded min_rd_win_size.
1	rd_priority	'1' gives reads priority when soft_win_ena = '1' if wr_win has exceeded min_wr_win_size.
0	soft_win_ena	'1' enables soft windows.

4.5.13 IRM Queue State RAM

The Queue State RAM can only be accessed from the MMC when the IRM is disabled (ic_enable = '0' in IRM_CONTROL). It is accessed as follows:

Writes:
1. Write target QSR address to IRM_QSR_ADDR register.
2. Write to IRM_QSR_DATA0-3 in order. Writes to IRM_QSR_DATA0-2 are latched and held until a write to IRM_QSR_DATA3 occurs, at which point all 100 bits of data are written to the QSR line pointed to by IRM_QSR_ADDR.

Reads:
1. Write target QSR address to IRM_QSR_ADDR register.
2. Read IRM_QSR_DATA0-3 in any order.

4.5.13.1 IRM Q State RAM address (IRM_QSR_ADDR)

Bits	Field Name	Description
31:6	-	Not used
5:0	qsr_addr	Address for accessing Q state RAM

4.5.14 IRM Q State RAM data 0 (IRM_QSR_DATA0)

Bits	Field Name	Description
31	aempty_level[0]	Bit 0 of almost empty threshold value, in qwords
30:20	afull_level[10:0]	Almost full threshold value, in qwords
19:10	frm_cnt[9:0]	Number of complete frames in queue
9:0	cur_frm_qw_cnt[9:0]	Qwords received since SOP for current frame

4.5.14.1 IRM Q State RAM data 1 (IRM_QSR_DATA1)

Bits	Field Name	Description
31:21	rd_pointer[10:0]	Current read pointer
20:10	queue_level[10:0]	Current queue level, in qwords
9:0	aempty_level[10:1]	Bits 10:1 of almost empty threshold value, in qwords

4.5.14.2 IRM Q State RAM data 2 (IRM_QSR_DATA2)

Bits	Field Name	Description
31:22	first_line[9:0]	Bits 9:0 of memory offset of first line of this port queue
21:11	last_line[10:0]	Memory offset of last line of this port queue
10:0	wr_pointer[10:0]	Current write pointer

4.5.14.3 IRM Q State RAM data 3 (IRM_QSR_DATA3)

Bits	Field Name	Description
31:4	-	Not used
3	cut_thru_state	'1' = This port queue is in cut-through mode
2	dump_til_eop_state	'1' = All incoming traffic for this port will be discarded thru the next end-of-frame
1	port_valid	'1' = this port is active. Must be set to '0' for inactive/unconfigured ports.
0	first_line[10]	Bit 10 of memory offset of first line of this port queue

4.5.15 IRM Cut-Through Timeout RAM

The Cut-Through Timeout RAM (CTR) can only be accessed from the MMC when the IRM is disabled (ic_enable = '0' in IRM_CONTROL). It is accessed as follows:

Writes:
1. Write target CTR address to IRM_CTR_ADDR register.
2. Write to IRM_CTR_DATA.

Reads:
1. Write target CTR address to IRM_CTR_ADDR register.
2. Read IRM_CTR_DATA.

4.5.15.1 IRM Cut-Through Timeout RAM address (IRM_CTR_ADDR)

Bits	Field Name	Description
31:6	-	Not used
5:0	ctr_addr	Address for accessing Cut-Through Timeout RAM

4.5.15.2 IRM Cut-Through Timeout RAM data (IRM_CTR_DATA)

Bits	Field Name	Description
31:8	-	Not used
7:0	ct_timeout_val	The cut-through timeout period for this port, in ticks of the master cut-through timer. The

Bits	Field Name	Description
		master cut-through timer tick period is set in the IRM_CT_TIMER_INC register.

4.5.16 IRM CRC Enable RAM

The CRC Enable RAM (CER) can only be accessed from the MMC when the IRM is disabled (ic_enable = '0' in IRM_CONTROL). It is accessed as follows:

Writes:
1. Write target CER address to IRM_CER_ADDR register.
2. Write to IRM_CER_DATA.

Reads:
1. Write target CER address to IRM_CER_ADDR register.
2. Read IRM_CER_DATA.

4.5.16.1 IRM CRC Enable RAM address (IRM_CER_ADDR)

Bits	Field Name	Description
31:3	-	Not used
2:0	cer_addr	W/R Address for accessing CRC Enable RAM

4.5.16.2 IRM CRC Enable RAM data (IRM_CER_DATA)

Bits	Field Name	Description
31:0	crc_ena[32*cer_addr+31: 32*cer_addr]	Each location in the CRC Enable RAM contains the crc_ena bits for 32 ports. Location 0 contains ports 31:0, location 1 contains ports 63:32, etc.

4.5.17 IRM Reassembly Buffer RAM

The Reassembly Buffer RAM (RB) can only be accessed from the MMC when the IRM is disabled (ic_enable = '0' in IRM_CONTROL). It is accessed as follows:

Writes:
1. Write target RB address to IRM_RB_ADDR register.
2. Write to IRM_RB_DATA0-4 in order. Writes to IRM_RB_DATA0-3 are latched and held until a write to IRM_RB_DATA4 occurs, at which point all 135 bits of data are written to the RB line pointed to by IRM_RB_ADDR.

Reads:
1. Write target RB address to IRM_RB_ADDR register.
2. Read IRM_RB_DATA0-4 in any order.

4.5.17.1 IRM Reassembly Buffer RAM address (IRM_RB_ADDR)

Bits	Field Name	Description
31:11	-	Not used
10:0	rb_addr	W/R Address for accessing Reassembly Buffer RAM

4.5.17.2 IRM Reassembly Buffer RAM data 0 (IRM_RB_DATA0)

Bits	Field Name	Description
31:0	rb_data[31:0]	W/R data when accessing Reassembly Buffer RAM

4.5.17.3 IRM Reassembly Buffer RAM data 1 (IRM_RB_DATA1)

Bits	Field Name	Description

Bits	Field Name	Description
31:0	rb_data[63:32]	W/R data when accessing Reassembly Buffer RAM

4.5.17.4 IRM Reassembly Buffer RAM data 2 (IRM_RB_DATA2)

Bits	Field Name	Description
31:0	rb_data[95:64]	W/R data when accessing Reassembly Buffer RAM

4.5.17.5 IRM Reassembly Buffer RAM data 3 (IRM_RB_DATA3)

Bits	Field Name	Description
31:0	rb_data[127:96]	W/R data when accessing Reassembly Buffer RAM

4.5.17.6 IRM Reassembly Buffer RAM data 4 (IRM_RB_DATA4)

Bits	Field Name	Description
31:7	-	Not used
6	rb_start	Frame start flag
5	rb_end	Frame end flag
4	rb_error	Frame error flag
3:0	rb_rd_size	Valid bytes in last qword of frame

4.5.18 IRM Rx Frame Counter Register (RX_FRM_CNT)

Bits	Field Name	Description
31:0	rx_frm_cnt	IRM Received frame counter counts EOPs received from the SPI-4 interface.

4.5.19 IRM Rx Byte Counter

The Rx Byte counter counts total bytes received from the SPI-4 interface. The counter is 38 bits wide so two accesses are required to read it. Read the counter as follows:

1. Read RX_BYTE_CNTR_L. Reading this location latches all 38 bits of the counter, clears the counter, and returns the lower 32 bits of the latched count value.
2. Read RX_BYTE_CNTR_H. Reading this location returns the upper 6 bits of the count value latched by the prior read of RX_BYTE_CNTR_L.

4.5.19.1 IRM Rx Byte Counter L Register (IRM_RX_BYTE_CNT_L)

Bits	Field Name	Description
31:0	irm_rx_byte_cnt_l	Lower 32 bits of Rx byte counter.

4.5.19.2 IRM Rx Byte Counter H Register (IRM_RX_BYTE_CNT_H)

Bits	Field Name	Description
31:6	-	Not used
5:0	irm_rx_byte_cnt_h	Upper 6 bits of Rx byte counter.

4.5.20 IRM Fwd Frame Counter Register (IRM_FWD_FRM_CNT)

Bits	Field Name	Description
31:0	irm_fwd_frm_cnt	IRM forwarded frame counter counts EOPs forwarded to Scratchpad from IRM.

4.5.21 IRM Fwd Byte Counter

The Forwarded Byte counter counts total bytes forwarded to the Scratchpad. The counter is 38 bits wide so two accesses are required to read it. Read the counter as follows:

1. Read FWD_BYTE_CNTR_L. Reading this location latches all 38 bits of the counter, clears the counter, and returns the lower 32 bits of the latched count value.
2. Read FWD_BYTE_CNTR_H. Reading this location returns the upper 6 bits of the count value latched by the prior read of FWD_BYTE_CNTR_L.

4.5.21.1 IRM Fwd Byte Counter L Register (IRM_FWD_BYTE_CNT_L)

Bits	Field Name	Description
31:0	irm_fwd_byte_cnt_l	Lower 32 bits of Fwd byte counter.

4.5.21.2 IRM Fwd Byte Counter H Register (IRM_FWD_BYTE_CNT_H)

Bits	Field Name	Description
31:6	-	Not used
5:0	irm_fwd_byte_cnt_h	Upper 6 bits of Fwd byte counter.

4.5.22 IRM Runt Frame Counter Register (IRM_RUNT_FRM_CNT)

Bits	Field Name	Description
31:16	irm_if_runt_frm_cnt	Counts frames <= 16 bytes dropped by the IRM Input Filter module.
15:0	irm_ic_runt_frm_cnt	Counts frame >16 bytes but <MinFrameSize dropped by the IRM Input Controller module.

4.5.23 IRM Jabber and Cut-Thru Frame Counter Register (IRM_JABR_CT_FRM_CNT)

Bits	Field Name	Description
31:16	irm_jabber_frm_cnt	Counts frames >MaxFrmSize detected and truncated by IRM Input Controller module.
15:0	irm_ct_frm_cnt	Counts frames that are forwarded with the IRM Input Controller in cut-through mode.

4.5.24 IRM Dropped QWord Counter Register (IRM_DROPD_QW_CNT)

This counter counts qwords dropped in three different places in the IRM. This count is not a 100% accurate indication of data dropped for two reasons:

1. Each dropped qword may contain anywhere from 1-16 bytes. The counter does not indicate the "fullness" of each qword.
2. Two of the three drop events counted are mutually exclusive (can not occur at the same time), but the third drop event can happen at the same time as the other two. When this occurs only one dropped qw is counted.

Bits	Field Name	Description
31:0	irm_dropped_qw_cnt	Counts 16-byte quad-words dropped by the IRM.

4.5.25 IRM Read Command Queue Write Register (IRM_RCQ_WRITE)

Bits	Field Name	Description
31:9	-	Not Used
8	rcq_dump_flag	When '1', next frame will be dumped.
7:0	rcq_port_num	Port number of next frame to be read from Reassembly Buffer.

4.5.26 Frame Forwarder Control (FF_CONTROL)

Bits	Field Name	Description
31:1	-	Not Used
0	ff_enable	Module enable for frame forward module (1 = enabled). Reset state = 0.

4.5.27 Frame Forwarder Status (FF_STATUS)

Bits	Field Name	Description
31:12	-	Not Used
11	module_idle	'1' = Frame Forwarder is in idle state
10	crc_fifo_empty	'1' = CRC output FIFO s empty
9	sp_full	'1' = Scratchpad is applying backpressure to IPU
8	lp0_rdy	'1' = LP0 is ready to accept new frame
7	lp1_rdy	'1' = LP1 is ready to accept new frame
6:3	event_seq_num	Event sequence number
2:0	frm_state	Frame forwarder state

4.5.28 Frame Forwarder PIT RAM

The Port Information Table (PIT) RAM can only be accessed when the Frame Forwarder is disabled (ff_enable = '0' in FF_CONTROL). It is accessed as follows:

Writes:
1. Write target PIT address to FF_PIT_ADDR register.
2. Write to FF_PIT_DATA.

Reads:
1. Write target PIT address to FF_PIT_ADDR register.
2. Read FF_PIT_DATA.

4.5.28.1 Frame Forwarder PIT RAM Address (FF_PIT_ADDR)

Bits	Field Name	Description
31:8	-	Not used
7:0	PIT_addr	PIT access address through MMC when module is disabled

4.5.28.2 Frame Forwarder PIT RAM Data (FF_PIT_DATA)

Bits	Field Name	Description
31:15	-	Not used
14:12	PP_CODE_OFFSET	PP code offset. Part of side info sent to IPM with each frame header.
11:0	VLAN_ID	VLAN_ID. Part of side info sent to IPM with each frame header.

4.5.29 CRC Payload Start Reference (CRC_PYLD_ST_REF)

Bits	Field Name	Description
31:6	-	Not used
5:0	payld_start_ref	Payload start reference. Indicates the byte offset from the start of the frame (0 = first byte after CRC header) of the reference point for the payload byte offset fields in the CRC header. The CRC payload starts payld_start_ref + payload_byte_offset bytes from the start of the frame.

4.5.30 Debug Mux Control (IPU_DEBUG_MUX)

Bits	Field Name	Description
31:3	-	Not used
2:0	debug_mux_sel	Mux select control for debug output mux

4.5.31 IPU Interrupt Register (IPU_INT)

Bits	Field Name	Description
31:8	-	Not used

Bits	Field Name	Description
7	ff_err	Frame Forwarder detected illegally framed data
6	crc_missing_eop_err	CRC Engine detected missing EOP (and forced an EOP with the frame error bit set).
5	crc_hdr_err	CRC engine stalled waiting for a payload to end after EOP arrived.
4	misc_config_err	Illegal IPU top-level register access
3	ipm_config_err	Illegal IPM register access
2	pp_config_err	Illegal PP register access
1	ff_config_err	Illegal Frame Forwarder register access
0	irm_config_err	Illegal IRM register access

4.5.32 IPU Interrupt Enable Register (IPU_INT_ENA)

Bits	Field Name	Description
31:8	-	Not used
7	ff_err_ena	HIGH to enable Frame Forwarder error interrupt.
6	crc_missing_eop_err_ena	HIGH to enable CRC Engine missing EOP error interrupt.
5	crc_hdr_err_ena	HIGH to enable CRC Engine CRC header error interrupt.
4	misc_config_err_ena	HIGH to enable miscellaneous illegal MMC access error interrupt.
3	ipm_config_err_ena	HIGH to enable IPM illegal MMC access error interrupt.
2	pp_config_err_ena	HIGH to enable PP illegal MMC access error interrupt.
1	ff_config_err_ena	HIGH to enable Frame Forwarder illegal MMC access error interrupt.
0	irm_config_err_ena	HIGH to enable IRM illegal MMC access error interrupt.

4.6 Initialization

The following sequence should be used to initialize the IPU after reset.

1. Complete SPI-4 module initialization (see SPI-4 Module HLD).
2. Release IPU module reset.
3. Initialize Packet Processor external configuration registers.
4. Complete Packet Processor initialization, but keep LP0 and LP1 stop bits asserted (see PP HLD).
5. Initialize IRM Queue State RAM.
6. Initialize IRM Cut-Through Timeout (CTR) RAM.
7. Initialize IRM CRC Enable (CER) RAM.
8. Initialize Frame Forwarder Port Information Table (PIT) RAM.
9. Enable Input Event Queue and Input Header Buffers 0 and 1 (IPM_CONTROL).
10. Clear LP0 and LP1 stop bits to enable PP (see PP HLD).
11. Enable Frame Forwarder (FF_CONTROL).
12. Set CRC Engine payload start reference (CRC_PYLD_START_REF).
13. Initialize IRM control register and enable the IRM Input Controller (IRM_CONTROL).

4.7 IPU Diagnostic Support Features

4.7.1 Internal Traffic Generation Using the IRM

The IPU allows a processor, via the MMC, to fill the Reassembly Buffer with frames and schedule those frames to be forwarded by writing to the Read Command Queue. This mechanism allows up to 32K bytes of real-time traffic to be forwarded from the IPU to the Scratchpad and Dispatcher with no external SPI-4 traffic source. The procedure to use this mechanism is as follows:

1. Determine the number of SPI-4 ports, the port queue sizes, and the IPU mode (full-frame or normal) desired.
2. Perform steps 2-12 of the IPU initialization procedure in section 4.6).

3. Write test traffic to the IPU Reassembly Buffer, keeping in mind the port queue boundaries established in step 2. Keep track of the number of frames, total qwords, and write pointer settings within each port queue.
4. Update the Queue State RAM entry for each port queue to reflect the current state of the queue (wr_pointer = one location past last location written; queue_level = # of qwords in queue; frm_cnt = # of frames in queue).
5. Set the order in which the frames should be forwarded by writing the port number of each frame to the Read Command Queue in the desired forwarding order. The Read Command Queue must contain one entry for each frame to be forwarded. Set the rcq_dump_flag to '0' for all entries.
6. Complete step 13 the IPU initialization procedure (ensure that the full_frame_mode flag is set to the correct state).

At this point the IPU will process and forward normally all frames that were scheduled to be forwarded in the Read Command Queue. When the Read Command Queue is empty, the IPU will stop. If desired, external traffic could then be processed normally, but there is no mechanism to generate internal traffic continuously. The IPU Input Controller must be disabled and the above procedure repeated to generate additional internal traffic. However, the IPU IRM can be reconfigured independently of the rest of the IPU and Pegasus (see the warm reconfiguration procedure in section 3.1.7), so continuous traffic of sorts can be generated in bursts, albeit with large gaps of time between bursts.

4.7.2 Internal Traffic Generation Using the Packet Processor

The Packet Processor could be programmed to generate Input Events continuously with no header input from the IPU. For this to work, both PP Logical Processors must be programmed to generate events because the IPU Input Event Queue module has a fixed event forwarding order that alternates between Logic Processors. If a single event is generated it must come from LP0.

4.7.3 Memory Access

All IPU internal non-FIFO memories can be completely read and written via the MMC. The Read Command Queue FIFO can be written via the MMC but cannot be read. MMC memory access is only possible when the module in which the memory resides is disabled (see register map for details). Memory access allows a processor to inspect memory contents after the IPU has been halted to support system debug.

5 Design Rationale

5.1 Read Command Queue Sizing

The Read Command Queue must be able to hold as many read commands as the Reassembly Buffer can hold frames. The Input Filter module discards all frames <= 16 bytes (one qword), so the minimum frame size that can be written to the Reassembly Buffer is two qwords. The Reassembly Buffer can hold 2048 qwords, so it can hold a maximum of 1024 minimum-size frames. Therefore, the Read Command Queue must be 1024 entries deep to ensure that overflow is not possible.

5.2 Reassembly Buffer Read/Write Access Timing

An original design goal for the IPU was to share the Reassembly Buffer RAM between reads and writes on an every-other-cycle basis. This goal proved unreasonably complex to implement, so instead time was split into multi-cycle read and write "windows". The size of these windows is configurable up to 32 cycles each. Although this simplified the IPU design, it had the adverse effect of making Reassembly Buffer throughput bursty. Since the rest of the IPU datapath, including the SPI-4 input, runs continuously, several FIFOs had to be added to prevent Reassembly Buffer burstiness from impacting other IPU modules.

6 Open Issues/Action Items

None.

7 Summary

The preceding sections should have accurately described the operation of the Input Processing Unit. Please notify the author of any discrepancies, omissions, or typos.